VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Ladění fyzického návrhu databáze v SQL Serveru

Automated Physical Database Design and Tuning in SQL Server

2015

Petr Stodůlka

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Petr Stodůlka**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Ladění fyzického návrhu databáze v SQL Serveru
Automated Physical Database Design and Tuning in SQL Server

Zásady pro vypracování:

V rámci této práce se student seznámí s nástrojem pro automatické ladění fyzického návrhu v SQL Serveru a otestuje jeho možnosti ve srovnání s intuitivním návrhem provedeným člověkem. Prvním cílem práce je vytvoření aplikace, která umožní otestovat a porovnat více fyzických návrhů pro jednu databázi. Druhým cílem je vyzkoušení různých variant fyzického návrhu pro TPC-H benchmark a jejich porovnání s návrhem nalezeným nástrojem pro automatické ladění fyzického návrhu v SQL Serveru.
Úkoly:
1. Seznámení se s technikami používanými pro automatické ladění fyzického návrhu databáze.
2. Analýza, návrh a implementace aplikace pro otestování a ohodnocení varianty fyzického návrhu.
3. Příprava TPC-H benchmarku a jeho ruční a automatické odladění.
4. Porovnání těchto benchmarků s pomocí nástroje vytvořeném v bodě dva.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Radim Bača, Ph.D.**

Datum zadání: 01.09.2014

Datum odevzdání: 07.05.2015

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2015 .....

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli, protože bez nich by tato práce nevznikla. Práce byla podpořena grantem č. SGS SP2014/211: Škálovatelnost databázových systémů.

Abstrakt

Tato diplomová práce pojednává o ladění fyzického návrhu databáze v SQL Serveru. SQL Server má k dispozici nástroj Database Engine Tuning Advisor, sloužící pro nalezení optimálního fyzického návrhu. Jedním z cílů této práce je ukázat, že intuitivní návrh člověka bude často lepší než návrh, který nalezne Database Engine Tuning Advisor. Dalším cílem bude vytvoření aplikace, která umožní otestovat a porovnat více fyzických návrhů pro databázi. Nakonec provedeme porovnání fyzických návrhů pro TPC-H Benchmark databázi, které byly získány jak nástrojem SQL Serveru, tak člověkem.

Klíčová slova: MS SQL Server, fyzický návrh, plán vykonání dotazu, pohled, indexovaný pohled, TPC-H Benchmark, IS DB Shepherd

Abstract

This master's thesis is about automated physical database design and tuning in SQL Server. SQL Server has a Database Engine Tuning Advisor tool used to find the optimal physical database design. One of the goals of this work is to show that the intuitive physical database design built by man will often be better than the design found by Database Engine Tuning Advisor. Another goal will be to create an application for testing and comparing of other physical database designs. Finally, we will compare the TPC-H Benchmark database designs created by both the SQL Server tool and the human designer.

Keywords: MS SQL Server, physical design, execution plan, view, indexed view, TPC-H Benchmark, IS DB Shepherd

Seznam použitých zkratek a symbolů

SŘBD	–	systém řízení báze dat
Query Optimizer	–	optimalizátor dotazu
Execution Plan	–	plán vykonání dotazu
Database engine	–	jádro databázového stroje
Open-source	–	program s volně dostupným zdrojovým kódem
OLTP	–	Online transaction processing
TPC	–	Transaction Processing Performance Council

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod do problematiky ladění databáze

Ladění databáze je dosti důležitým aspektem v oblasti informačních systémů, jelikož efektivita databázového systému je důležitá pro efektivitu veškerých navazujících operací. Někdy se systém prohlásí za příliš pomalý a pořídí se nový, příliš nákladný hardware i přes to, že se neprovedlo řádné ladění databáze. Ladění lze provést za pomocí nástroje pro nalezení automatického fyzického návrhu, nebo také i manuálně nalezením vlastního fyzického návrhu.

Fyzickým návrhem databáze máme na mysli specifikaci, která obsahuje datové struktury jako indexy, shlukované indexy, indexované pohledy atd. Tyto datové struktury nám mohou urychlit odezvu provádění SQL příkazů. Ovšem při nesprávném použití mohou dobu provádění navýšit, proto je třeba porozumět jejich výhodám i nevýhodám.

Nástroj SQL Serveru Database Engine Tuning Advisor umožnuje pro dané vytížení získat doporučený fyzický návrh databáze. Dokonale pracuje s indexy, ale hůře už s indexovanými pohledy. V této práci se pokusíme nalézt efektivnější vlastní fyzický návrh oproti doporučenému fyzickému návrhu. Nejprve je ale třeba porozumět problematice při vytváření samotného fyzického návrhu. Rovněž musíme nasimulovat zatížení, které se bude co nejvíce blížit reálnému prostředí. V poslední řadě je zapotřebí odchytit a porovnat naměřené výsledky jak pro vlastní fyzický návrh, tak doporučený fyzický návrh. Součástí této práce bude i vytvoření aplikace, která umožní otestovat a porovnat více fyzických návrhů databáze.

Ve druhé kapitole si popíšeme vykonávání dotazu a také se seznámíme s optimalizátorem dotazu. Třetí kapitola pojednává o problematice ladění databáze, kde si vysvětlíme komplexitu vytváření fyzických návrhů. Ve čtvrté kapitole budou charakterizovány jednotlivé datové struktury, které lze při vytváření fyzického návrhu využít. Kapitola pátá pojednává o TPC-H Benchmarku, který jsme využili při testování. Šestá kapitola popisuje vytvořenou aplikaci pro testování a porovnání fyzických návrhů. V poslední sedmé kapitole si zobrazíme výsledné naměřené výsledky.

2 Vykonávání dotazu

Ladění databází úzce souvisí s komponentou každého SŘBD a tou je *optimalizátor dotazu*. Samotný optimalizátor je u většiny SŘBD obdobný, ovšem v této práci se zaměříme pouze na SQL Server optimalizátor.

2.1 SQL Server optimalizátor dotazu

Jedná se o cenově založený optimalizátor, který analyzuje množinu *plánů vykonání dotazu* pro daný dotaz, odhaduje cenu těchto plánů a vybírá ten s nejnižší cenou. Samozřejmě optimalizátor není schopen uvažovat veškeré možné plány pro každý dotaz, jelikož musí brát v potaz časovou náročnost vyhledání potenciálních plánů a odhadnutí ceny těchto plánů.

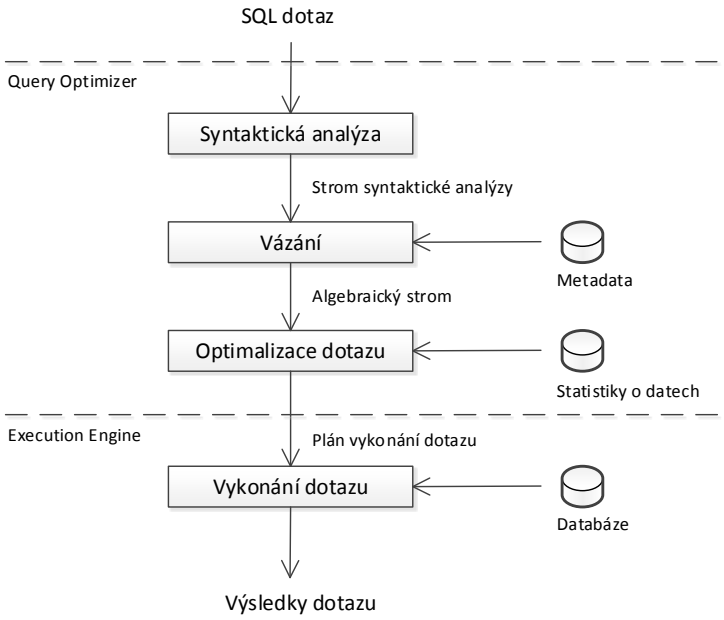
Jedná se o komponentu SQL Serveru, která má největší dopad na výkon databáze. Koneckonců vybrání správného/špatného plánu může způsobit rozdíl času vykonání dotazu v milisekundách, minutách či dokonce hodinách.

2.1.1 Query Processor

Jádro SQL Serveru představují dvě hlavní komponenty: *storage engine* a *query processor*, také nazývaný jako *relational engine*. Storage engine je zodpovědný za čtení dat mezi diskem a samotnou pamětí. A to takovým způsobem, že se optimalizuje souběžnost a zachovává se integrita dat. Naproti tomu query processor provádí veškeré dotazy podané na SQL Server, sestaví plán pro jejich optimální vykonání, provede jej a navrátí požadované výsledky.

Jednotlivé dotazy jsou předány SQL Serveru jako příkazy jazyka SQL (nebo také T-SQL, což je procedurální rozšíření jazyka SQL od Microsoft SQL Serveru). Jelikož je SQL vysokoúrovňový deklarativní jazyk, definujeme pouze, jaká data mají být získána z databáze, nikoliv kroky k jejich získání. Query processor má za úkol sestavit požadovaný plán pro každý obdržený dotaz tak rychle, jak je to jen možné. Dalším úkolem je vykonání dotazu nad daným plánem.

Query processor disponuje samostatnou komponentou *query optimizer*, která provádí cenovou optimalizaci plánu vykonání dotazu. Sestaví plán a pak jej předá další komponentě a tou je *execution engine*, který plán vykoná a získá výsledky z databáze. K nalezení nejlepšího plánu pro vykonání dotazu, query processor provádí řadu kroků, které jsou schematicky zobrazeny na obrázku 1.



Obrázek 1: Query processor proces

- Syntaktická analýza - syntaktická analýza nám zajišťuje validní syntaxi T-SQL dotazů a překládá SQL dotaz do počáteční stromové reprezentace. Každý uzel stromu reprezentuje logickou operaci, kterou musí dotaz vykonat, ať už jde o čtení konkrétní tabulky nebo vykonání vnitřního spojení. Zpočátku budou tyto logické operace úzce spjaty s původní syntaxí dotazu;

- Vázání - vázání je nejčastěji spjato s rozlišováním názvů. Během operace vázání se SQL Server ujišťuje, zdali veškeré názvy objektů skutečně existují a zdali odpovídají veškeré názvy tabulek a sloupců stromu syntaktické analýzy s korespondujícím objektem v systémovém katalogu. Kontroluje se také, zdali odpovídají jednotlivé datové typy. Výstupem tohoto druhého procesu je tzv. algebraický strom;

- Optimalizace dotazu - Optimalizace dotazu je proces mapování logických operací dotazu na operace fyzické. Originální logické operace jsou přetransformovány na operace fyzické, které je schopen execution engine vykonat. Některé operace jako je například třízení, se přetransformují na stejné fyzické operace, naproti tomu jiné logické operace se mohou přetransformovat na několik možných fyzických operací. Například logická operace spojení může být mapována na fyzické operátory, jako jsou *nested loops join*, *merge join*, nebo *hash join*;

- Generování možných plánů vykonání dotazu - za pomocí logického stromu, query optimizer sestaví řadu možných způsobů, jak vykonat dotaz, respektive řadu možných plánů vykonání dotazu;

- Stanovení ceny každého plánu - query optimizer odhaduje cenu pro každý vygenerovaný plán. Plán, jehož odhadovaná cena je nejnižší ze všech těchto plánů, je předán komponentě execution engine;
- Vykonání dotazu - execution engine vykoná dotaz podle zvoleného plánu. Samotný plán může být uložen v paměti, konkrétně v *plan cache*.

3 Problematika ladění databáze

Relační databázové systémy zajišťují fyzickou nezávislost dat. Nezáleží na použitém fyzickém návrhu databáze [2], dotazy vždy vracejí stejné výsledky. Ovšem z hlediska výkonu, můžeme fyzickým návrhem velmi značně ovlivnit dobu provádění dotazů.

Neexistuje věc, jako je optimální fyzický návrh pro databázi a dané vytížení, kdy je zapotřebí mít k dispozici jak databázi, tak i vytížení. Důvodem je, že indexy jsou vhodné, pouze pokud zrychlují dotazy bez značného zpomalení aktualizací. Vyhovující indexy pro dané vytížení mohou být pro jiné vytížení nevýznamné či dokonce neblahé. Navíc indexy jsou redundantní struktury zabírající diskové úložiště. Jelikož je úložiště vždy limitováno, rozhodnutí, které indexy mají být součástí fyzického návrhu databáze, se stává obtížným problémem. Problém nalezení fyzického návrhu může být neformálně definován následovně: Na vstupu máme mez uložiště (*bound B*) a dané vytížení (*workload W*) obsahující dotazy a aktualizace. Výstup nám tvoří získaný fyzický návrh (*configuration C*), který splňuje *B* a výsledky dotazů z *W* se vykonají tak efektivně, jak jen je to možné.

První otázkou je, co přesně tvoří vytížení *W* a jak je získáno. Nejčastěji uvažujeme vytížení jako sadu SQL dotazů a aktualizací s možností přidělení váhy (důležitosti) ke každému dotazu z vytížení. Vytížení může být vygenerováno po monitorování produkčního databázového systému po nějakou dobu, a to vykonáváním statických analýz databázových aplikací nebo také po pečlivém ručním návrhu. Požadujeme pouze takové reálné vytížení *W*, které je schopen systém řízení báze dat (SŘBD) vykonat.

Druhou otázkou je, jak stanovit metriku pro vybrání nejlepšího plánu vykonání dotazu ze všech možných. Tady se opět dostáváme k problému: "vykonání dotazů ve *W*, tak efektivně, jak jen je to možné". To vyžaduje měření aktuálních časů vykonání dotazů a aktualizací (tyto mohou způsobit změny v databázovém systému), k vyhodnocení daného fyzického návrhu nebo ke srovnání dvou fyzických návrhů. A navíc je vyžadováno, aby systém vykonávající dotazy nebyl ovlivňován jinými souběžnými databázovými dotazy nebo dokonce procesy operačního systému. Jelikož je tohoto v praxi skoro nemožné dosáhnout, je obtížné uvažovat takovouto metriku, kdy rovněž časy vykonání jsou obvykle nedeterministické. Alternativní přístup, který nevyžaduje vykonávání dotazů, je modelování ceny dotazu pod daným fyzickým návrhem. Pokud model přesně předpovídá čas vykonání dotazu bez jeho skutečného spuštění, může sloužit jako náhrada pro efektivní metriku a k vyřešení veškerých předešlých problémů.

Poslední otázkou je, jak seskupit jednotlivé ceny dotazu. Nejjednodušší a nejpoužívanější metrika sečte odhadovanou cenu všech dotazů a tím stanovuje výslednou cenu vytížení. Alternativní metriky zahrnují minimalizaci ceny nejpomalejšího dotazu z vytížení nebo průměrný poměr zlepšení.

Nyní si můžeme uvést formální definici problému nalezení fyzického návrhu, obsahující veškeré aspekty popsané výše:

Problém nalezení fyzického návrhu: Na vstupu máme dané vytížení $W = \{Q_1, \ldots, Q_n\}$, kde každý Q_i představuje SQL příkaz a mez uložiště B. Výstup nám tvoří získaný

fyzický návrh C= {I_1, ...,I_k}, kdy $\sum_j size(I_j) \leq B$ a zároveň $\sum_j$ *cost(Q_j, C)* je minimalizována, kde *cost(Q,C)* je cena optimálního plánu pro Q nalezené optimalizátorem, kdy veškeré a pouze indexy z C jsou k dispozici.

3.1 Komplexita fyzického návrhu databáze

Pro vyjasnění komplexity fyzického návrhu [2] využijeme TPC-H Benchmark 5, jenž modeluje dotazy střední složitosti. Konkrétně dotaz č. 5 reprezentující výpis 1, který obsahuje spojení přes 6 tabulek, podmínky zahrnující vestavěné funkce jako je *dateadd*, klauzuli *group by* a aritmetické operace v klauzuli *SELECT*.

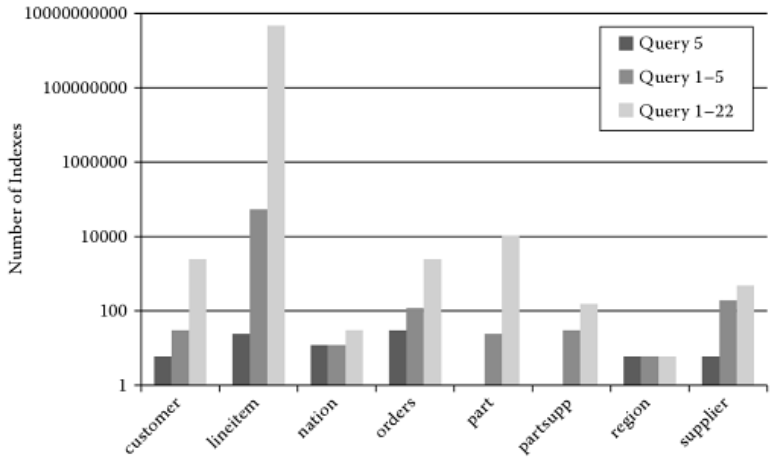
```
SELECT N_NAME,
   SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS REVENUE
FROM CUSTOMER,
   ORDERS,
   LINEITEM,
   SUPPLIER,
   NATION,
   REGION
WHERE C_CUSTKEY = O_CUSTKEY AND
   L_ORDERKEY = O_ORDERKEY AND
   L_SUPPKEY = S_SUPPKEY AND
   C_NATIONKEY = S_NATIONKEY AND
   S_NATIONKEY = N_NATIONKEY AND
   N_REGIONKEY = R_REGIONKEY AND
   R_NAME    = 'ASIA' AND
   O_ORDERDATE >= '1994-01-01' AND
   O_ORDERDATE < DATEADD(YY, 1, cast ('1994-01-01'as date))
GROUP BY N_NAME
ORDER BY REVENUE DESC;
```

Výpis 1: Dotaz střední složitosti

Získávání fyzického návrhu pro vytížení obsahující řadu takovýchto dotazů je velmi složité. Odkážeme se na knihu od autora Nicolas Bruno [2], kde se provedl výpočet počtu možných indexů, které mohou být obsaženy v plánech vykonání dotazu optimalizátoru. Obrázek 2 zobrazuje tento počet pro každou tabulku z TPCH databáze a odlišné vytížení. Například, pokud budeme uvažovat pouze dotaz č. 5, dostaneme kolem 80 možných indexů. Ať se nám toto číslo může zdát malé, naším úkolem je vybrat vhodnou podmnožinu (fyzický návrh) z těchto 80 možných indexů s exponenciální složitostí. Pokud bychom si vzali prvních pět dotazů pro TPC-H Benchmark, dostali bychom přes 50 000 možných indexů. A nakonec pro vytížení obsahující všech 22 dotazů, dostáváme přes miliardu možných indexů. Očividně, uvažování s tak velkým počtem indexů je velmi obtížným úkolem.

3.2 Automatizovaný fyzický návrh

V předchozí podkapitole jsme si ukázali, že i při středně velkém problému je velmi obtížné manuálně nalézt optimální fyzický návrh databáze pro dané vytížení. A proto může



Obrázek 2: Počet možných indexů pro odlišné vytížení [2].

být výhodné použití automatizovaných nástrojů [2]. Jelikož je problém fyzického návrhu NP-úplný, je zde malá naděje na nalezení optimálního fyzického návrhu automatizovanými nástroji, ale nalezený fyzický návrh se bude blížit optimálnímu.

Problém fyzického návrhu se může zdát jako komplexní vyhledávací problém. Můžeme jej popsat a analyzovat z hlediska tří komponent: vyhledávací prostor fyzického návrhu, cenový model k ohodnocení fyzického návrhu a výpočetní strategie napříč vyhledávacím prostorem.

3.2.1 Vyhledávací prostor

Jedná se o množinu možných fyzických návrhů, s nimiž může být uvažováno jako s potencionálními kandidáty pro danou databázi a vytížení.

Jednou z charakterizací vyhledávajícího prostoru, kterou si nyní popíšeme je velmi obecná a nezávislá na aktuálním vytížení. Zpočátku uvažujeme, že možné indexy jsou pouze jedno-sloupcové, z důvodu jednoduchosti a také faktu, že většina vytížení byla vygenerována jednoduchými OLTP [1] aplikacemi, kde dotazy jsou spíše jednoduššího rázu (většinou navigační dotazy, kde je vybrán pouze jediný záznam, s možností spojení s další tabulkou k získání dodatečných informací). Postupem času se přišlo na to, že se stává přístup s možnými jedno-sloupcovými indexy nedostatečný. Nynější aplikace pro podporu rozhodování často generují komplexní dotazy, kdy možné jedno-sloupcové indexy často selžou za účelem zvýšení výkonu. Řešením je rozšíření vyhledávajícího prostoru o možné více-sloupcové indexy.

Další charakterizací vyhledávajícího prostoru je rovněž nezávislá na vytížení, kdy se předpokládá použití jednoho indexu na jednu tabulku.

Po vybrání dané charakterizace vyhledávajícího prostoru, ještě musíme rozhodnout jaký bude *specifický* vyhledávací prostor pro dané vytížení. Nechceme totiž jako součástí

[1] http://en.wikipedia.org/wiki/Online_transaction_processing

vyhledávajícího prostoru takové indexy, které se zaručují být neužitečné (indexy nad tabulkami, které se nevyskytují ve vytížení). Tradiční způsob definování vyhledávajícího prostoru pro dané vytížení se skládá ze dvou kroků. V prvé řadě se jedná o postup definující množinu možných indexů pro každý dotaz vytížení. Jsou ovšem k dispozici i jiné techniky. Druhým krokem je definování vyhledávajícího prostoru pro vytížení. Některé z původních technik omezují takovýto prostor jednoduše svázáním indexů pro každý dotaz. Naproti tomu novější přístupy také uvažují o přídavných indexech, které vzniknou kombinací možných indexů z jiných dotazů.

3.2.2 Cenový model

K provedení jakéhokoliv smysluplného průchodu vyhledávacím prostorem fyzického návrhu, je zapotřebí spočítání velikosti jakéhokoliv fyzického návrhu (configuratin) *C* a také ceny ohodnocení vytížení pod fyzickým návrhem *C*. Odhadování velikosti fyzického návrhu není nijak složité, jelikož je úměrná součtu velikostí všech indexů ve fyzickém návrhu. Může být spočítána na základě statistik odpovídající bázové tabulky (počet záznamů a velikost záznamu).

Výpočet cenového ohodnocení vytížení pod libovolným fyzickým návrhem je větší problém. Nejvíce precizní postup zahrnuje vytváření indexů ve fyzickém návrhu, optimalizaci veškerých dotazů z vytížení a přidání odhadované ceny pro všechny dotazy. Jelikož je optimalizace dotazů relativně levná záležitost, vytváření indexů ve fyzickém návrhu je nákladnější, z důvodu opakovatelného skenování a třídění tabulek v databázi. Tento problém řeší implementace *what-if optimalization layer* obsahující odhadovanou výpočetní cenu plánů pod libovolným fyzickým návrhem bez nutnosti zhmotnění indexů.

3.2.3 Výpočetní strategie

Dřívější výpočetní strategie byly založeny bud'to na zjednodušeném vyhledávajícím prostoru (pouze jedno-sloupcové indexy) nebo omezeném výpočetním prostředí (OLTP dotazy s externím cenovým modelem). Z tohoto důvodu některé prostředky považují za problém získávání optimálního fyzického návrhu pro dané vytížení. Z důvodu komplexních výsledků v této oblasti, ale také i narůstajících sofistikovaných dotazovacích nástrojů a vytížení, současné techniky využívají heuristiku k efektivnímu přechodu na menší, ale relevantní části vyhledávajícího prostoru.

4 Ladění databáze

V této kapitole si popíšeme jednotlivé techniky, které lze uplatnit při ladění databáze. Zaměříme se na popis několika datových struktur a jejich vzájemnou kombinaci. Budou zde popsány pouze struktury pro SQL Server, na který je tato práce zaměřena. Některé z použitých datových struktur byly převzaty z mé bakalářské práce [3].

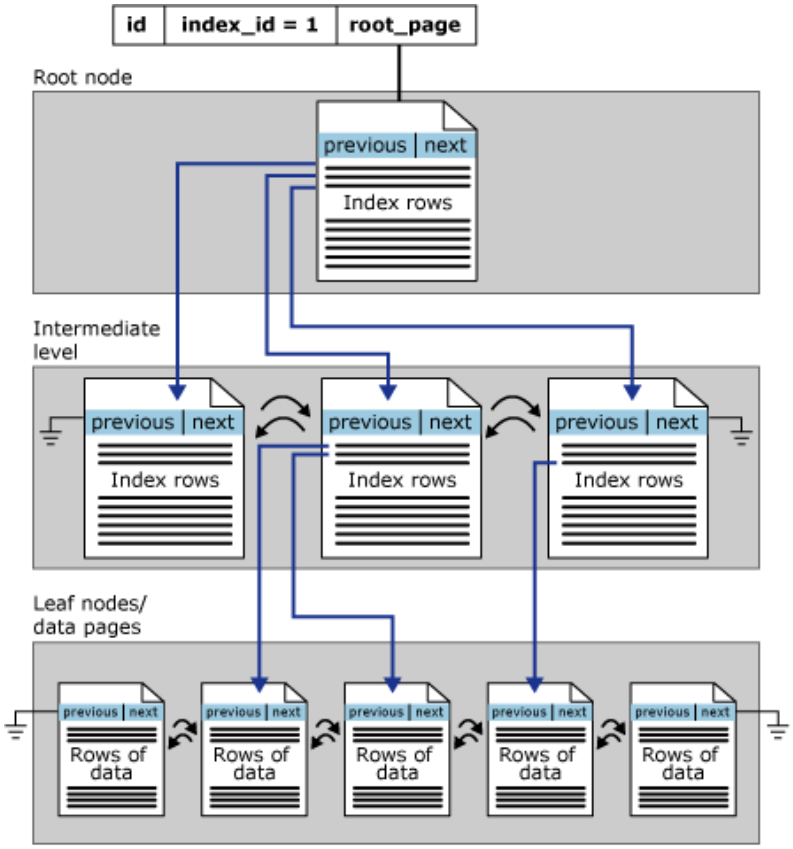
4.1 Pravidla při vytváření fyzického návrhu databáze

V odborných literaturách [2] můžeme nalézt tato pravidla pod pojmem *rules of thumb*. Postupem času se objevily určité formy obecných pokynů těchto pravidel a osvědčené postupy během vytváření fyzických návrhů. Některé z nich jsou následující:

- *Indexace primárních klíčů a hlavně cizích klíčů.* Jelikož se spojení ve většině případů provádí mezi sloupci primárních a cizích klíčů, indexy zahrnující tyto sloupce mohou vést ke zlepšení výkonu u komplexnějších dotazů.
- *Sloupce často se vyskytující v klauzuli WHERE jsou dobrými kandidáty na index.* Sloupce, které jsou odkazovány v podmínce rovnosti či nerovnosti v klauzuli *WHERE* (zvláště ty, na které je často odkazováno) mohou být efektivně využity příslušnými indexy.
- *Vyhnout se redundantním indexům.* Indexy definované nad stejnými (nebo skoro stejnými) sloupci jsou zřídka užitečné, jelikož neposkytují případné výhody navíc. Zároveň využívají více úložného prostoru a také musí být tyto indexy aktualizovány.
- *Použití indexovaných sloupců vzhledem k jejich časté aktualizaci.* Pokud je databázový sloupec aktualizován, veškeré indexy definované nad tímto sloupcem jsou rovněž aktualizovány. Sloupec, který je často aktualizován, může způsobit zhoršení výkonu u jinak užitečného indexu.
- *Zvážení pokrývajících indexů u kritických dotazů.* Pokrývající indexy (indexy obsahující veškeré požadované sloupce dotazu) jsou velmi vhodné pro zlepšení výkonu, ale zároveň mohou být příliš velké a užitečné pouze pro malé množství dotazů. Ovšem dotazy, které jsou kritické nebo velmi časté, mohou efektivně využívat výhodu pokrývajících indexů.
- *Vyhnout se indexům nad malými tabulkami.* Indexy nad tabulkami, které se vlezou do jedné či dvou stránek obvykle neposkytují případné vylepšení výkonu. Pouze navyšují počet struktur k údržbě a administraci. Důvodem je, že optimalizátor raději prochází řádky tabulky sekvenčně, což se mu zdá jako výhodnější řešení.

4.2 Clustered index

Tento typ indexu [4] třídí a ukládá záznamy v tabulce na základě jejich klíčové hodnoty. Můžeme jej vytvořit pouze jeden v rámci tabulky, jelikož záznamy samotné mohou být setříděny pouze v jednom pořadí.



Obrázek 3: Struktura Clustered indexu [4]

V SQL Serveru jsou indexy organizovány do struktury B-Stromu. U indexu typu B-Strom je každá stránka nazývána tzv. ”index node“ neboli indexový uzel. Horní uzel B-Stromu se nazývá kořenový uzel. Naproti tomu spodní úroveň uzlů indexu nazýváme listové uzly. Jakékoliv úrovně indexu mezi kořenem a listovými uzly jsou označovány jako střední úrovně. U clustered indexu listové uzly obsahují stránky příslušné tabulky. Kořen a uzly střední úrovně obsahují stránky indexu držící řádky indexu. Každý řádek indexu obsahuje klíčovou hodnotu a ukazatel, a to buď na stránku střední úrovně v B-Stromu, nebo na řádek v listové úrovni indexu. Stránky každé úrovně indexu jsou propojeny tzv. dvojitým propojeným seznamem.

Clustered indexy můžeme naleznout v pohledu sys.partitions. Defaultně má Clustered index jediný ”partition“ neboli segment. Pokud má ovšem více segmentů, tak každý segment má vlastní B-Stromovou strukturu obsahující data pro specifický segment. Pokud by měl index 4 segmenty, tak bychom měli 4 B-Stromové struktury, každá ve svém segmentu.

Strukturu cluster indexu v jednom segmentu znázorňuje obrázek 3.

4.3 Nonclustered index

Nonclustered indexy [5] mají stejnou B-Stromovou strukturu jako cluster indexy, až na následující významné rozdíly:

- Řádky příslušné tabulky nejsou setříděny a uloženy na základě jejich nonclustered klíče.
- Listy nonclustered indexu jsou tvořeny ze stránek indexu nikoliv ze stránek tabulky.

Noncluster index může být definován na tabulce nebo pohledu s clustered indexem nebo také na tabulce typu halda. Každý řádek indexu obsahuje nonclustered klíčovou hodnotu a řádkový lokátor. Tento lokátor ukazuje na řádek v clustered indexu nebo tabulce typu halda mající klíčovou hodnotu.

Řádkový lokátor v řádcích nonclustered indexu je buď to ukazatel na řádek tabulky, nebo seskupený indexový klíč pro daný řádek tabulky, jak je popsáno níže:

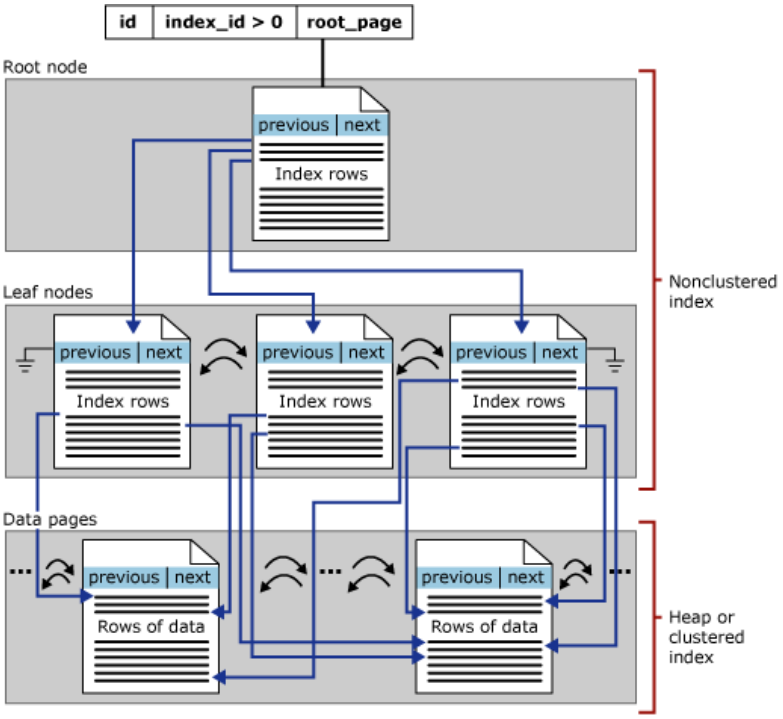
- Pokud se jedná o tabulku typu halda 4.4, což znamená, že nedisponuje cluster indexem, řádkový lokátor je ukazatel na řádek tabulky. Ukazatel je sestaven z identifikátoru souboru (ID), čísla stránky a počtu řádků na stránce. Tento ukazatel je znám jako Row ID (RID).
- Pokud tabulka obsahuje cluster index nebo je přítomen index na indexovaném pohledu, tak řádkový lokátor je klíčová hodnota cluster indexu pro daný řádek. Pokud cluster index není unikátní index, SQL Server vytvoří několik duplikátních unikátních klíčů přidáním interně generované hodnoty zvané uniquefier. Tato čtyř bytová hodnota není viditelná pro uživatele. Je přidána pouze tehdy, když se vyskytne požadavek na vytvoření unikátního clustered klíče pro použití v nonclustered indexech. SQL Server vyhledává řádek tabulky vyhledáním cluster indexu s využitím cluster index klíče uloženého v listu řádku noncluster indexu.

Strukturu Nonclustered indexu v jednom segmentu znázorňuje obrázek 4.

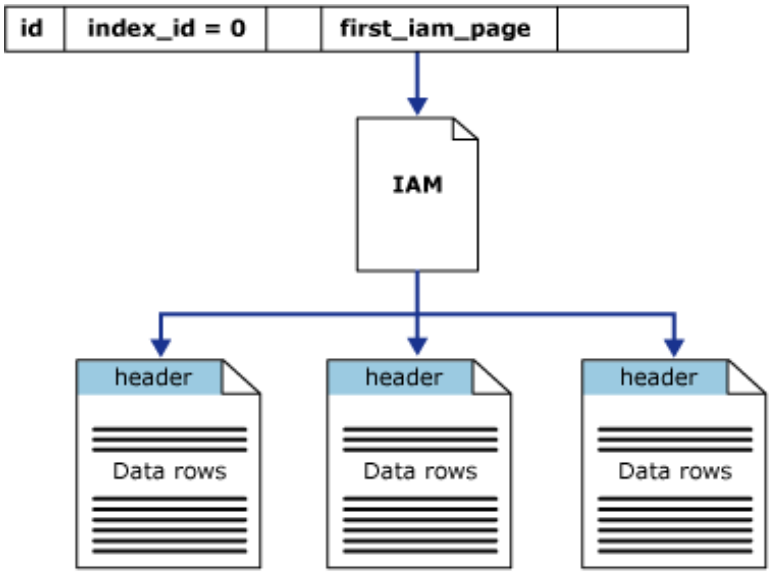
4.4 Heap table (tabulka typu halda)

Nesmíme také zapomenout na tabulku typu halda, jedná se o standardní databázovou tabulku [6]. Po vytvoření nejsou na ni aplikovány žádné indexy s výjimkou primárních klíčů. Ovšem nad touto tabulkou můžeme později vytvořit indexy. Data jsou zde uloženy bez specifického pořadí. Obvykle jsou zpočátku data uloženy v takovém pořadí, jak byly vloženy do tabulky, ale databázový engine může přesouvat data k efektivnějšímu ukládání řádků. Tím pádem nelze předvídat pořadí dat. Pro specifikaci pořadí ukládání řádků, je nutné vytvořit cluster index nad touto tabulkou, tím pádem tabulka už nebude typu halda.

Tento typ tabulky je výhodný v případě velkého počtu operací vkládání. Pokud nad tabulkou nebudou vytvořeny žádné nonclustered indexy, pak musí být procházena celá



Obrázek 4: Struktura Nonclustered indexu [5]



Obrázek 5: Struktura Heap table [6]

tabulka (operace TABLE SCAN) k nalezení patřičného záznamu. Toto může být přijatelné, pokud se bude jednat o tabulku s malým počtem záznamů. U tohoto typu tabulky jsou jednotlivé řádky identifikovány pomocí reference na řádkový identifikátor (RID) obsahující číslo souboru, číslo stránky a slot na stránce. Jedná se o malou efektivní strukturu. V některých případech se tabulka typu halda používá, když jsou data vždy přístupná prostřednictvím nonclustered indexů a RID je menší než clustered index klíče.

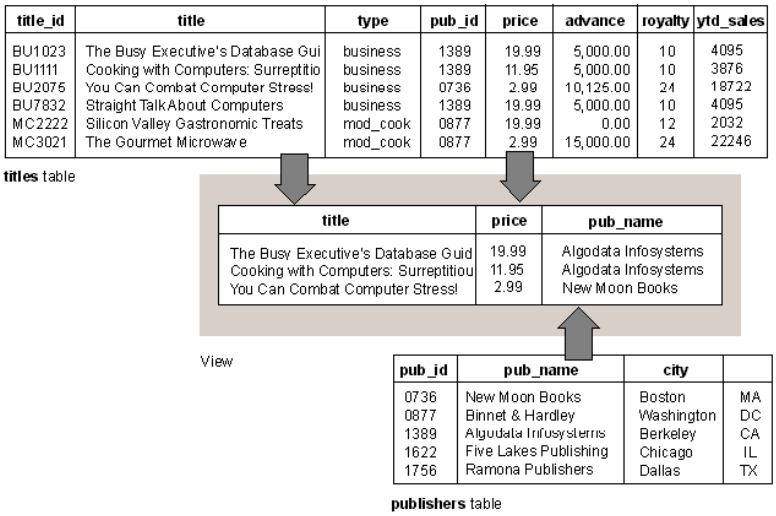
Tento typ tabulky není vhodný, pokud se jedná o rozsáhlou tabulku bez přítomnosti indexů nad touto tabulkou. Pro nalezení daného záznamu tabulky, musí dojít k přečtení veškerých záznamů tabulky.

Strukturu tabulky typu halda znázorňuje obrázek 5.

4.5 View

View neboli pohled [7] můžeme chápat jako virtuální tabulku, kdy obsah takovéto tabulky je dán dotazem. Pohled se chová jako normální tabulka, obsahuje jak sloupce, tak i záznamy. Ovšem pohled není fyzicky uložen jako sada datových hodnot v databázi. Jednotlivé sloupce a záznamy jsou odvozeny od tabulek, které jsou odkazovány dotazem při vytváření samotného pohledu. Navíc jsou zkonstruovány dynamicky při odkazu na pohled.

Pohled se chová jako filtr na tabulky, které jsou odkazované na samotný pohled. Dotaz který definuje pohled, se může pojit k jedné nebo více tabulek nebo dokonce k několika pohledům samotným ve stejné či jiné databázi. Distribuované dotazy mohou být také použity pro definici pohledů, které používají data z několika různorodých zdrojů.



Obrázek 6: Pohled založený na dvojici tabulek [7]

Tohle může být například vhodné pokud chceme kombinovat podobnou strukturu dat z různých serverů, kdy každý ze serverů ukládá data pro odlišnou část naší organizace.

Nejsou zde žádná omezení na vytvoření dotazů na pohled a existuje několik málo omezení pro modifikaci dat. Důležité ale je, že pohledy se nepoužívají z hlediska výkonového. Slouží především ke skrytí komplexity dotazu. Pokud budeme chtít navýšit výkon, mluvíme o tzv. Indexovaných pohledech neboli Index Views 4.6.

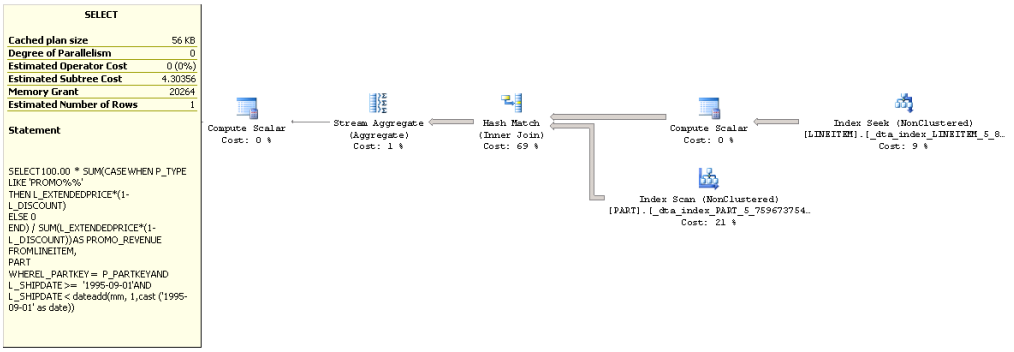
Obrázek 6 zobrazuje pohled založený na dvojici tabulek. Příkaz pro vytvoření tohoto pohledu nalezneme ve výpisu 2.

```
CREATE VIEW MyView
AS
SELECT t.title,
    t.price,
    p.pub_name
FROM titles t  JOIN publishers p on p.pub_id = t.pub_id
WHERE t.type = 'business';
```

Výpis 2: Vytvoření pohledu

4.6 Indexed View

Indexovaný pohled [8] se na rozdíl od běžného pohledu liší tím, že jej vytváříme za účelem navýšení výkonu. Získáme jej vytvořením unique cluster indexu na daný pohled. Po jeho vytvoření, se pohled tvářící se jako virtuální tabulka, fyzicky uloží na disk. Má svou vlastní stránkovou strukturu a přistupujeme k němu jako k normální tabulce. Indexovaný pohled může být mocným nástrojem za účelem navýšení výkonu, ale musíme být velice obezřetní. Pokaždé když provádíme modifikaci dat v tabulkách, které jsou vázány



Obrázek 7: Plán vykonání dotazu před optimalizací [8]

na pohled, tak se modifikují data jak v samotných tabulkách, tak i v indexech navázaných na pohled. Takovýto případ nám může ovlivnit výkon operace zápisu.

V úvodu této práce jsem se zmínil, že jednou z možností, jak by se dal automatický fyzický návrh vytvořený Database Engine Tuning Advisor překonat, je právě indexovaným pohledem. A proto zde uvedu příklad jeho vytvoření.

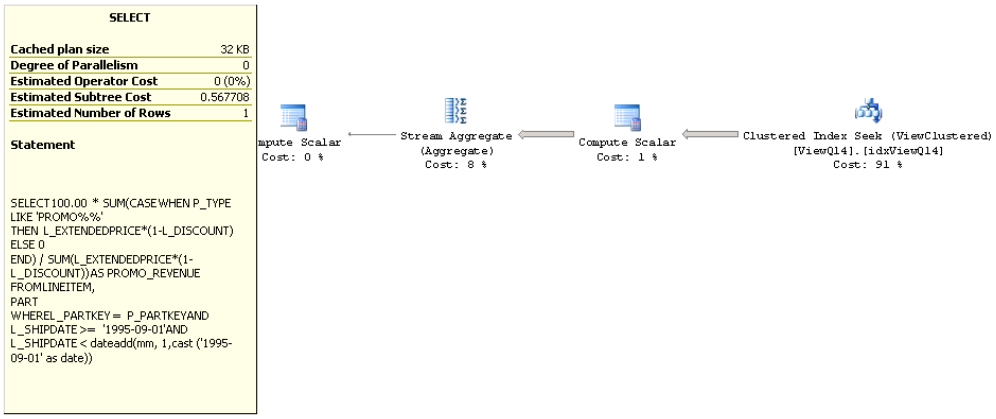
Vezmeme si dotaz z TPC-H Benchmarku, konkrétně se jedná o dotaz č. 14, jež nám monitoruje reakci trhu na propagaci, jako jsou televizní reklamy nebo speciální kampaně. Tento dotaz nalezneme v podkapitole 5.2 a samotný dotaz ve výpisu 3.

```
SELECT 100.00 * SUM (   CASE WHEN P_TYPE LIKE 'PROMO%%'
                        THEN L_EXTENDEDPRICE*(1-L_DISCOUNT)
                        ELSE 0
                    END) / SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS PROMO_REVENUE
FROM LINEITEM,
    PART
WHERE L_PARTKEY = P_PARTKEY AND
    L_SHIPDATE >= '1995-09-01'  AND
    L_SHIPDATE < dateadd(mm, 1,cast ('1995-09-01' as date))
```

Výpis 3: Dotaz č.14 TPC-H Benchmark

Plán vykonání dotazu by vypadal následovně, obrázek 7. Všimněme si hodnoty *Estimate Subtree Cost*, která nám reprezentuje počet logických čtení. Tuto hodnotu se budeme snažit co nejvíce snížit v rámci optimalizace.

V prvé řadě je potřeba vytvořit klasický pohled, nalezneme ve výpisu 4. Zde je při vytváření použita klauzule *WITH SCHEMABINDING*. Tato klauzule nám zaručuje navázání pohledu na schéma, ve kterém se nachází tabulky v pohledu. Není možné měnit schéma tabulek z důvodu ovlivněni definice pohledu. Jedinou možností je změnit definici pohledu nebo jeho smazání. Pokud použijeme klauzuli *WITH SCHEMABINDING*, musíme uvést ke každé tabulce, pohledu či funkci referenci, na jaké schéma se váže, tedy *schema_name.object_name*. Nutnou podmínkou je, že veškeré odkazované objekty se musí nacházet ve stejné databázi.



Obrázek 8: Plán vykonání dotazu po optimalizaci [8]

```
CREATE VIEW [dbo].[ViewQ14] WITH SCHEMABINDING AS
    SELECT L_ORDERKEY,
        L_PARTKEY,
        L_SHIPDATE,
        P_TYPE,
        L_EXTENDEDPRICE,
        L_DISCOUNT
    FROM dbo.LINEITEM,
        dbo.PART
    WHERE L_PARTKEY = P_PARTKEY
```

Výpis 4: Vytvoření pohledu pro indexovaný pohled

Vytvořením pohledu ovšem dotaz nijak neoptimalizujeme, počet logických čtení se nijak nezmění. V tomto okamžiku jsme pouze skryli komplexitu dotazu. Abychom optimalizovali tento dotaz, je zapotřebí vytvořit unique cluster index 4.2 na pohled, nalezneme ve výpisu 5.

```
CREATE UNIQUE CLUSTERED INDEX idxViewQ14
    ON [dbo].[ViewQ14] (L_SHIPDATE, L_ORDERKEY, L_PARTKEY)
```

Výpis 5: Vytvoření indexovaného pohledu

Při vytváření musíme splnit několik podmínek. V prvé řadě index musí být unikátní neboli *UNIQUE*. Dále sloupce, které mají být v indexu uloženy, se musí vyskytovat v klauzuli *SELECT*. Někdy se nám také může stát, že počet sloupců v indexu je nedostatečný, jelikož dojde ke snaze vložení duplicitních záznamů. To je však zcela nepřípustné z důvodu, že index musí být unikátní. V takovémto případě je třeba přidat další sloupce.

Po vytvoření indexu se nyní podívejme na plán vykonání dotazu, obrázek 8. Jak můžeme vidět, nyní již optimalizátor využívá vytvořený indexovaný pohled a je to znát na počtu logických čtení. Tento počet se snížil cca 8x.

Použití indexovaných pohledů nám opravdu zvýší výkon návrhu, ovšem musíme brát v potaz následující omezení [2]:

- Definice pohledu nemůže být vázána na pohledy, tabulky z jiné databáze než pohled samotný.
- Nemůže obsahovat funkce jako COUNT, MIN, MAX, outer joins, DISTINCT, TOP, ORDER BY, poddotazy atd.
- Nelze modifikovat základní tabulky a sloupce vázající se na pohled.
- Pohled musí být vytvořen s možností *WITH SCHEMABINDING*.
- Pokud definice pohledu obsahuje klauzuli *GROUP BY*, musí také obsahovat *COUNT_BIG(*)* a nesmí obsahovat *HAVING*. Také se musí klíč unique clustered indexu odkazovat pouze na sloupce uvedené v klauzuli *GROUP BY*.

[2] Omezení Indexed View - http://msdn.microsoft.com/en-us/library/ms191432.aspx

5 TPC-H Benchmark

Představuje výkonnostní test od organizace TPC [3]. Jedná se o neziskovou organizaci se zaměřením na definování databázových výkonnostních testů a šíření objektivních, ověřitelných údajů o výkonu pro průmysl. TPC-H Benchmark [9] je výkonnostní test podporující rozhodování. Skládá se ze sady obchodně orientovaných ad-hoc dotazů a souběžně modifikujících se dat. Dotazy a data sloužící k naplnění databáze, byly vybrány tak, aby pokryly široké odvětví průmyslu. Tento výkonnostní test ilustruje systémy pro podporu rozhodování, které zkoumají velké objemy dat, vykonávání dotazů s vysokým stupněm složitosti a získáváním odpovědí na rozhodující byznys otázky.

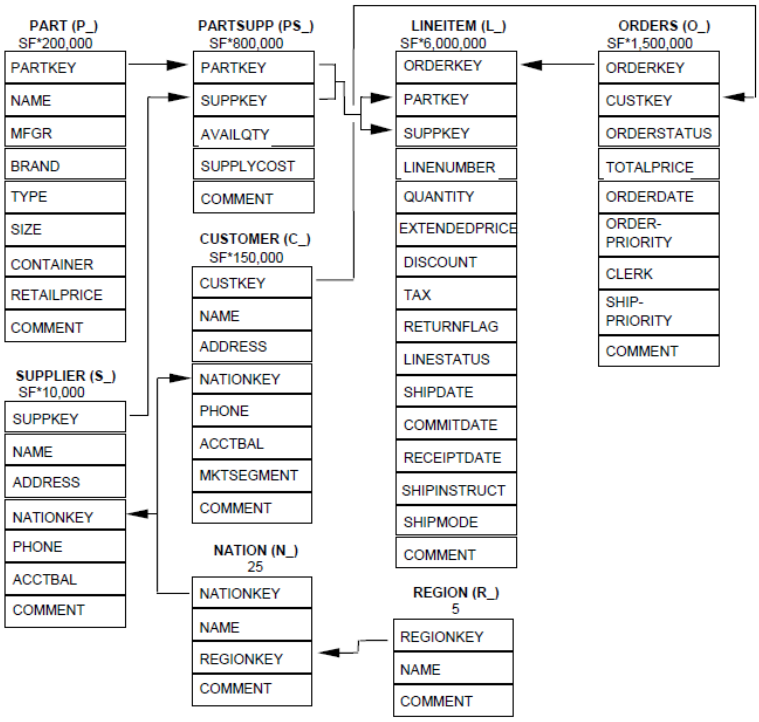
Výkonnostní metrika používaná TPC-H je měřena v počtu dotazů za hodinu a odráží několik aspektů. Těmito aspekty jsou:

- Stanovení velikosti databáze, vůči které jsou vykonávány dotazy.
- Výpočetní výkon dotazů, které jsou vykonávány v jednom sledu.
- Propustnost, kdy jsou dotazy získávány od vícero uživatelů zároveň.

5.1 TPC-H Schéma

Komponenty TPC-H výkonnostního testu se skládají z osmi jednotlivých tabulek (Bázových tabulek). Vztahy mezi sloupci těchto tabulek jsou ilustrovány na obrázku 9: TPC-H Schéma.

[3] Transaction Processing Performance Council - http://www.tpc.org/



Obrázek 9: TPC-H Schéma [9]

Legenda:

- V kulatých závorkách, následujících za každým jménem tabulky, jsou uvedeny prefixy pro veškeré sloupce dané tabulky.
- Šipky ukazují ve směru 1:N vztahy mezi tabulkami.
- Číslo nacházející se pod názvem tabulky reprezentuje kardinalitu neboli počet řádků tabulky. Některé z nich se pojí s rozsahovým faktorem, sloužící k získání možnosti zvolení velikosti databáze. Kardinalita pro tabulku LINEITEM je pouze přibližná.

5.2 TPC-H Dotazy

Tento výkonnostní test obsahuje sadu 22 dotazů a dvě databázové obnovující funkce, které musí být vykonány v rámci tohoto výkonnostního testu.

Každý dotaz je definován následujícími komponentami:

- **Byznys otázka**, která představuje byznys kontext, ve kterém by dotazy měly být použity.
- **Funkční definice dotazu**, která definuje použití jazyka SQL-92 k vykonání dotazu.

- **Substituované parametry**, které popisují, jak vygenerovat hodnoty potřebné ke zhotovení syntaxe dotazu.
- **Validate dotazů**, která popisuje, jak validovat dotazy vůči databázi.

5.2.1 Definice dotazů

V této části jsou popsány jednotlivé dotazy, i spolu s oběma obnovujícími databázovými funkcemi.

1. **Pricing Summary Report Query (Q1)**
   Tento dotaz reportuje množství zboží, které bylo zaúčtováno, dodáno a vráceno.
2. **Minimum Cost Supplier Query (Q2)**
   Tento dotaz vyhledá dodavatele, který by měl být vybrán pro objednání daného zboží v daném regionu.
3. **Shipping Priority Query (Q3)**
   Tento dotaz nalezne 10 kusů nedodaného zboží s nejvyšší cenou.
4. **Order Priority Checking Query (Q4)**
   Tento dotaz nám určuje, jak dobře funguje stanovení priorit objednávkového systému a stanoví nám výši uspokojení zákazníka.
5. **Local Supplier Volume Query (Q5)**
   Tento dotaz nám uvádí objem příjmů pro místní dodavatele.
6. **Forecasting Revenue Change Query (Q6)**
   Tento dotaz kvantifikuje množství nárůstu příjmů, které by mohly být výsledkem určitých slev v rámci společnosti v daném procentuálním rozsahu a v daném roce. Pomocí tohoto dotazu lze nalézt způsob, jak navýšit příjmy.
7. **Volume Shipping Query (Q7)**
   Tento dotaz stanovuje hodnotu zboží dodaného mezi určitými národy na pomoc při opětovném projednávání dodacích kontraktů.
8. **National Market Share Query (Q8)**
   Tento dotaz stanovuje, jak se změnil podíl na trhu v průběhu dvou let pro daný typ zboží v daném státu a v daném regionu.
9. **Product Type Profit Measure Query (Q9)**
   Tento dotaz určuje, jak velký profit je získán ze zboží ve spojení se státem a rokem dodavatele.
10. **Returned Item Reporting Query (Q10)**
    Tento dotaz identifikuje zákazníky, kteří by mohli mít problémy se zbožím, které jim bylo dodáno.

11. **Important Stock Identification Query (Q11)**
    Tento dotaz nám nalezne nejdůležitější podmnožinu dodavatelů v daném státě.

12. **Shipping Modes and Order Priority Query (Q12)**
    Tento dotaz zjišťuje, zdali méně nákladný způsob dodání negativně ovlivní důležité objednávky zákazníků tím, že zákazníci obdrží zboží po poslední možné lhůtě dodání.

13. **Customer Distribution Query (Q13)**
    Tento dotaz vyhledává vztahy mezi zákazníky a velikostí jejich objednávek.

14. **Promotion Effect Query (Q14)**
    Tento dotaz monitoruje reakci trhu na propagaci, jako jsou televizní reklamy nebo speciální kampaně.

15. **Top Supplier Query (Q15)**
    Tento dotaz stanovuje nejlepší dodavatele, aby mohli být oceněni, bylo jim poskytnuto více obchodních transakcí nebo byli identifikováni pro zvláštní uznání ve svém oboru.

16. **Parts/Supplier Relationship Query (Q16)**
    Tento dotaz zjišťuje, kolik dodavatelů může dodat zboží se specifickými vlastnostmi. Může být například použit pro stanovení, zdali je k dispozici vhodný počet dodavatelů pro obtížně objednatelné zboží.

17. **Small-Quantity-Order Revenue Query (Q17)**
    Tento dotaz stanovuje, jak by se průměrné roční příjmy změnily, pokud by se eliminovalo určité zboží s příliš malým počtem tohoto zboží na objednávkové listině. Tohle může vést ke snížení režijních nákladů a soustředit se spíše na prodej větších zásilek.

18. **Large Volume Customer Query (Q18)**
    Tento dotaz řadí zákazníky na základě jejich velikosti objednávek. Velké objednávky jsou definovány, jako ty objednávky, jejichž celkový počet zboží je nad určitým limitem.

19. **Discounted Revenue Query (Q19)**
    Reportuje hrubý diskontovaný příjem pojící se k danému zboží.

20. **Potential Part Promotion Query (Q20)**
    Identifikuje dodavatele ve specifickém státě, mající zboží, které by mohlo představovat kandidáta na propagační nabídky.

21. **Suppliers Who Kept Orders Waiting Query (Q21)**
    Dotaz identifikuje dodavatele, kteří nebyli schopni dodat zboží ve stanoveném termínu.

22. **Global Sales Opportunity Query (Q22)**
    Identifikuje lokalizaci zákazníků, kteří by byli ochotni koupit si nějaké zboží.

23. **New Sales Refresh Function (RF1)**
    Tato funkce přidává nové prodejní informace do databáze.

24. **Old Sales Refresh Function (RF2)**
    Tato funkce naopak odstraňuje staré prodejní informace z databáze.

6 IS DB Shepherd

Jedná se o informační systém umožňující otestování a porovnání více fyzických návrhů pro databázi. Má sloužit pro crowdsourcing databázové komunity, jehož výsledkem by bylo nalezení fyzických návrhů, které se velmi blíží optimálnímu návrhu.

Nyní si můžeme definovat základní pojmy z pohledu aplikace:

- Uživatelova databáze - vytváří samotný uživatel vložením logického schématu databáze, viz kapitola 6.3.3.
- Kontrolní/Testovací databáze - reálně vytvořená uživatelova databáze na reálném databázovém systému. Slouží pro přípravu všech ostatních částí uživatelovy databáze a také spouštění vytížení nad konkrétním fyzickým návrhem, viz kapitola 6.3.1.
- Vytížení - vytížení je tvořeno seznamem SQL dotazů. Spuštěním vytížení zatěžujeme uživatelovu databázi pro následné odchycení naměřených výsledků.
- Fyzický návrh - fyzickým návrhem databáze máme na mysli specifikaci, která obsahuje datové struktury, jakou jsou indexy, shlukované indexy, indexované pohledy atd. Tyto datové struktury nám mohou urychlit odezvu provádění SQL příkazů.
- Aplikační databáze - databáze, kde jsou uložena veškerá data. Jedná se o informace spjaté s uživatelovou databází, vytížením, fyzickým návrhem apod. Viz kapitola 6.5.

6.1 Specifikace požadavků

Tato podkapitola popisuje požadavky a specifikace systému, které vznikly jako součást této diplomové práce. Požadavky byly formulovány vedoucím této diplomové práce.

Podstatou systému je umožnit posouzení kvality fyzického návrhu pro určitou uživatelovu databázi a její vytížení. Uživatelova databáze bude mít dva stavy, kontrolní a testovací. Kontrolní stav bude sloužit pro přípravu všech potřebných částí uživatelovy databáze, zatímco v testovacím stavu bude docházet ke spouštění vytížení nad testovaným fyzickým návrhem.

K systému budou přistupovat dva typy uživatelů. V prvé řadě se bude jednat o uživatele, kterým bude umožněno vytvářet a udržovat uživatelovy databáze (dále jen tvůrci databází). Budou moci předpřipravovat uživatelovu databázi pro následné spuštění vytížení nad fyzickým návrhem. Příprava spočívá ve vytvoření samotné uživatelovy databáze, vložení dat pro její naplnění a také vložení vytížení. Jakmile je uživatelova databáze plně připravena k testování, převede se do stavu testování, teprve v tomto stavu se mohou vkládat fyzické návrhy. Tito uživatelé budou disponovat následujícími funkcemi:

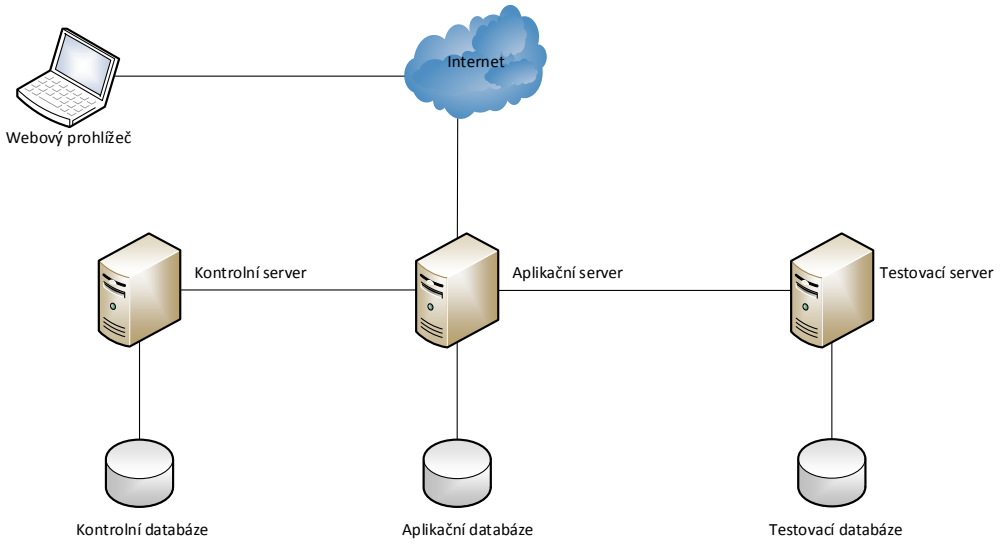
- Vkládání uživatelovy databáze - budou se zadávat tyto údaje: název, popis a maximální velikost uživatelovy databáze, dále pak výběr příslušného SŘBD, odpovídající ER diagram, DDL reprezentující logické schéma databáze a nakonec komentář k samotnému vložení.

- Přidání dat do uživatelovy databáze - data budou přidána v ZIP souboru obsahující soubory CSV, SQL nebo TBL. Data budou dvojího typu a to kontrolní a testovací. Kontrolní data budou omezena kapacitou do 10MB.
- Vkládání vytížení - vytížení může být v předdefinovaném XML formátu nebo jako množina SQL příkazů. Vytížení bude dvojího typu a to vytížení pro kontrolní data a vytížení pro testovací data.
- Převedení uživatelovy databáze do stavu testování - převedení uživatelovy databáze z kontrolního stavu do stavu testování, přičemž se zkontroluje konzistence jednotlivých částí uživatelovy databáze.
- Zobrazení seznamu vlastních uživatelových databází - bude zobrazeno: název, stav, SŘBD databáze, možnost stažení logického schématu a dat k odpovídající databázi.
- Zobrazení detailu jedné uživatelovy databáze - bude zobrazeno: název, stav, SŘBD databáze, datum vytvoření a poslední změny nad uživatelovou databází, možnost vložení vytížení a dat.

Dále se bude jednat o běžné uživatele, kteří disponují funkcemi:

- Registrace - budou se zadávat tyto údaje: login, jméno, příjmení, heslo a výběr zdali uživatel bude mít funkci tvůrce databáze (vytvářet a udržovat vlastní databáze - vytvoření databáze, vložení dat a vytížení) nebo běžného uživatele.
- Zobrazení vlastního profilu - budou zobrazeny údaje: login, jméno, příjmení, typ uživatele, počet uživatelových databází a fyzických návrhů.
- Zobrazení seznamu uživatelových databází ve stavu testování - bude zobrazeno: název, režim, SŘBD databáze, možnost stažení logického schématu a dat k odpovídající databázi.
- Zobrazeni detailu jedné uživatelovy databáze ve stavu testování - bude zobrazeno: popis uživatelovy databáze spolu s ER diagramem, možnost stáhnutí příslušných dat (logické schéma a data pro naplnění databáze).
- Vkládání fyzického návrhu - budou se zadávat tyto údaje: název, komentář fyzického návrhu a DDL obsahující příkazy k vytvoření fyzického návrhu.
- Zobrazení výsledků - a to jak výsledků pro všechny své vytvořené fyzické návrhy, tak i nejlepší výsledky ostatních uživatelů pro danou uživatelovu databázi.

Tvůrci databází budou rovněž disponovat funkcemi běžného uživatele. Jednotlivé funkce jsou zobrazeny Use Case diagramem na obrázku 22, který nalezneme v příloze A.



Obrázek 10: Diagram rozložení

6.2 Architektura řešení

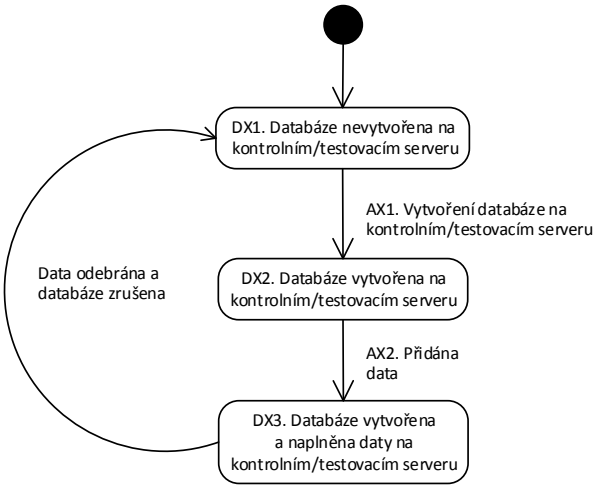
Systém využívá několik serverů, které mezi sebou komunikují pomocí protokolů HTTP a FTP. Rozložení těchto serverů je zachyceno pomocí diagramu rozložení na obrázku 10. Těmito servery jsou:

- Aplikační server - server, na němž běží samotná .NET aplikace. Data týkající se uživatelových databází, vytížení, fyzických návrhů apod. budou uložena v aplikační databázi, která bude reprezentována Microsoft SQL Server běžící na tomto serveru. Na aplikačním serveru budou také uloženy veškeré soubory typu: DDL, ZIP, SQL apod.
- Kontrolní databázový server - server, kde se spouštějí veškeré synchronní kontrolní úkoly. Zde budou uloženy DDL soubory spjaté s vytížením.
- Testovací databázový server - server, kde se spouštějí asynchronní testovací úlohy, jako je spouštění vytížení nad uživatelovou databází. I na tomto serveru budou uloženy DDL soubory spjaté s vytížením.

Na všech databázových serverech je nainstalovaný Microsoft SQL Server.

6.3 Dynamický náhled

V této podkapitole budou popsány jednotlivé funkce popisující chování systému a také zde bude zachyceno chování systému v časových návaznostech.



Obrázek 11: Stavy kontrolní/testovací databáze z pohledu kontrolního a testovacího serveru

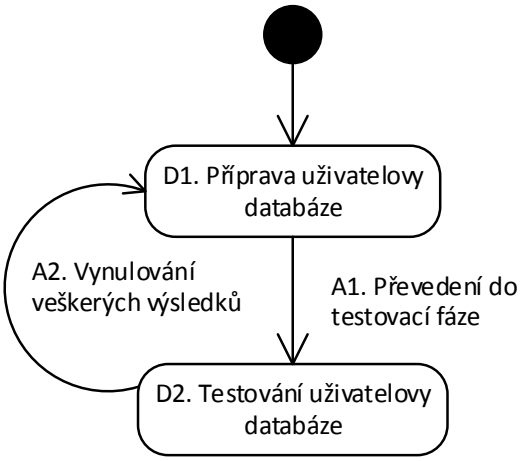
Jak jsme si již řekli, systém využívá několik serverů, kde kontrolní server zpravidla slouží k validaci (kontrola syntaxe jednotlivých příkazů, možnost vytvoření samotné uživatelovy databáze, korektnost přidaných dat pro uživatelovu databázi apod.). Naproti tomu testovací server slouží především ke spouštění vytížení nad konkrétním fyzickým návrhem a získávání naměřených výsledků.

6.3.1 Stavy kontrolní/testovací databáze

Kontrolní/testovací databáze se může nacházet ve třech stavech. Jedná se o reálně vytvořenou uživatelovou databázi na reálném databázovém systému. Na obrázku 11 se nachází stavový diagram z pohledu přítomnosti kontrolní/testovací databáze na kontrolním nebo testovacím serveru.

Přechody u stavů

- AX1, AX2 - Před vytvořením kontrolní/testovací databáze či vložením dat musí dojít k ověření, že na serveru je dostatek místa. Dostatek místa mimo jiné znamená, že na serveru je alespoň 1GB volného místa. Pokud není dostatek místa, musí se ze serveru vymazat nevyužité kontrolní/testovací databáze (na základě časových razítek). Bude docházet k mazání pouze těch kontrolních/testovacích databází, které obsahují nějaká data. Takto máme jistotu, že nesmažeme čerstvě vytvořené kontrolní/testovací databáze.



Obrázek 12: Životní cyklus uživatelovy databáze

6.3.2 Stavy uživatelovy databáze

Základní životní cyklus uživatelovy databáze je znázorněn aktivitním diagramem, viz obrázek 12. Po jejím vložení na aplikační server a vytvoření na kontrolním serveru se nachází ve stavu přípravy. Po vložení všech potřebných informací se uživatelova databáze přepne do stavu testování, ve kterém je možné vkládat fyzické návrhy. Stav uživatelovy databáze může ovlivnit provedení dalších kroků jako je vkládání dat, vytížení a fyzického návrhu.

Přechody u stavů

- A1 - Před převedením uživatelovy databáze do stavu testování, musí být vložena testovací data (musí také korektně projít kontrolou) a musí existovat vytížení pro testovací data, které prošlo kontrolou.
- A2 - Pokud dochází k převodu stavu z testování do přípravy (kontrolní stav), musí dojít k zneplatnění všech naměřených výsledku pro danou uživatelovu databázi. Uživatel je o této změně jasně informován a musí potvrdit tuto akci (Tato funkcionalita ještě není naimplementovaná).

6.3.3 Vkládání uživatelovy databáze

Vkládání uživatelovy databáze znázorňuje aktivitní diagram, viz obrázek 13. Nejprve dojde ke kontrole skriptu s logickým schématem za účelem nalezení syntaktických chyb.

Přeskočitelné chyby:

- Existují tabulky bez primárního klíče nebo bez vazby na jinou tabulku.
- Existuje tabulka T mající primární klíč X a jiná tabulka s atributem se stejným názvem X, který ale není cizím klíčem ukazujícím na tabulku T.
- Skript obsahuje definici, která je spojená s fyzickým návrhem (indexy, komprese atd.).

Nepřeskočitelné chyby:

- SQL příkazy s chybnou syntaxí pro vybrané SŘBD.
- Pokus o vytvoření uživatelovy databáze z DDL selhal na nějaké chybě.

Dále dojde k pokusu o vytvoření uživatelovy databáze na kontrolním serveru dle přiloženého DDL souboru. Tato akce slouží pouze pro kontrolu, zdali vůbec lze uživatelovu databázi vytvořit.

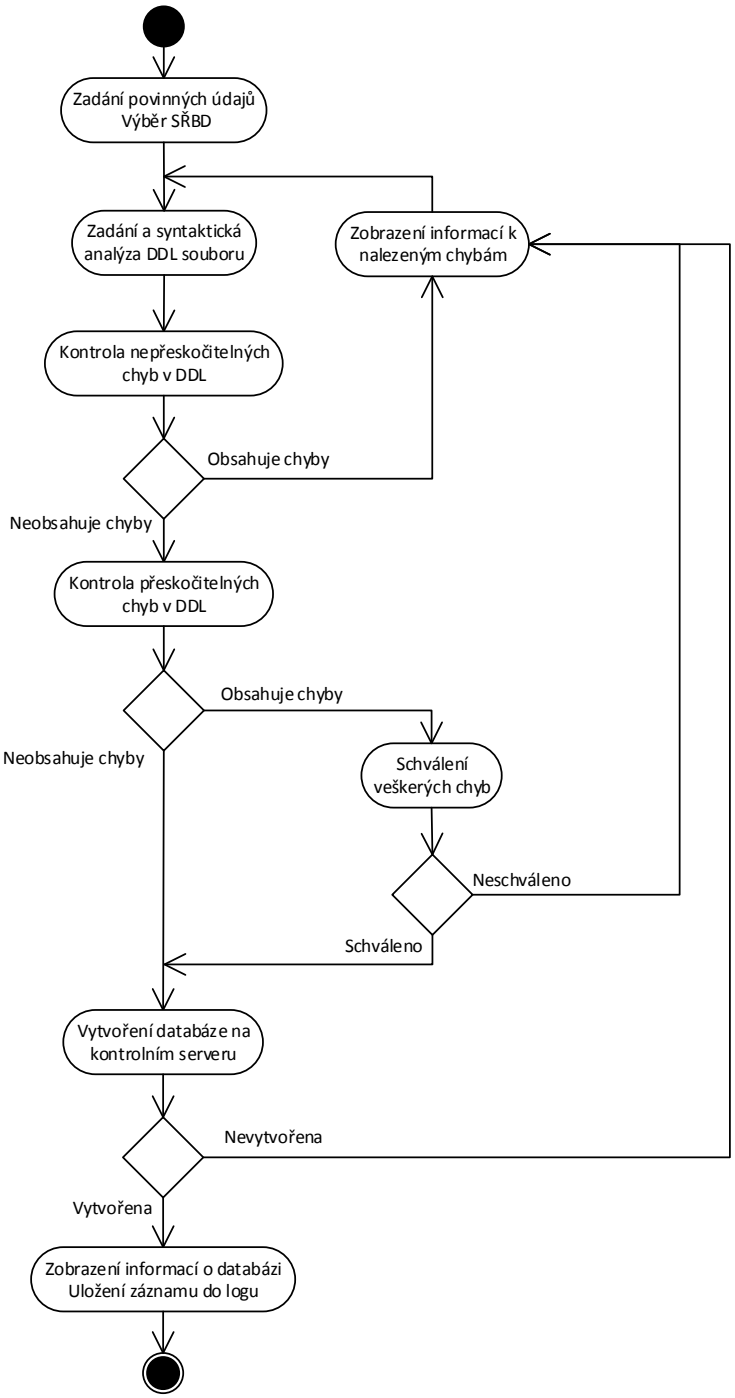
Podmínky pro vložení uživatelovy databáze:

- Název uživatelovy databáze musí být jedinečný, pokud již název existuje, je třeba upozornit uživatele o jeho existenci.
- DDL soubor musí být přijat pouze ve formátu SQL. Nesmí obsahovat DML příkazy jako je *INSERT*, *UPDATE*, *DELETE*. Naopak může obsahovat DDL příkazy, jako je vytváření indexů, pohledů apod.

6.3.4 Přidání dat pro uživatelovu databázi

Samotná data jsou uložena na aplikačním serveru. Data jsou dvojího typu, a to kontrolní a testovací. Kontrolní data slouží pouze pro kontrolu (jak samotných dat, tak i následné ověření vytížení a fyzického návrhu) a jsou tudíž kapacitně omezena do 10MB. Tato data slouží pro naplnění kontrolní/testovací databáze na kontrolním serveru. Naproti tomu testovací data nejsou nijak omezena a slouží pro naplnění kontrolní/testovací databáze na testovacím serveru, kde se bude spouštět vytížení. Samotné přidání dat je znázorněno aktivitním diagramem, viz obrázek 14.

Jak můžeme vyčíst z obrázku, přidání dat na kontrolní/testovací server se dosti liší. Při přidání kontrolních dat, se tato data vloží rovnou do kontrolní/testovací databáze. Naproti tomu u testovacích dat, dojde k jejich vložení později (není zobrazeno na obrázku). Zásadní věcí je, že pokud byla data přidána na aplikační server, musíme smazat veškeré vytížení pojící se na kontrolní/testovací databázi z důvodu uchování konzistence. Smažou se pouze vytížení odpovídající typu dat, tedy kontrolní vytížení nebo testovací vytížení, viz kapitola 6.3.5. Ke smazání dojde na několika místech najednou: v aplikační databázi, na aplikačním serveru a na kontrolním/testovacím serveru. Po úspěšném



Obrázek 13: Vkládání uživatelovy databáze

nahrání dat do kontrolní/testovací databáze, dojde k zálohování samotné kontrolní/testovací databáze pro rychlou obnovu původního stavu databáze po provedení vytížení.

Podmínky pro přidání dat:

- Data jsou přidávána pouze v zip souboru, který může obsahovat soubory typu CSV, TBL, SQL. Kdy SQL soubor může obsahovat pouze DML dotazy. CSV soubory musí obsahovat hlavičku s názvy sloupců. Daný soubor s daty se musí jmenovat jako název tabulky, pro kterou jsou data určena.
- Kontrolní data musí být přidána jako první, ještě před testovacími daty.
- Pokud kontrolní/testovací databáze není ve stavu DX2 (vytvořena na kontrolním/-testovacím serveru), pak se ji do toho stavu pokusíme dostat, viz obrázek 11.

6.3.5 Vkládání vytížení

Vytížení můžeme vkládat dvojího typu, a to vytížení pro kontrolní data a testovací data. Uživatel může vkládat vytížení, pouze zdali jsou přítomna odpovídající data. Povolené formáty souboru, ve kterých je vytížení uloženo, je SQL nebo XML. SQL soubor obsahuje pouze čisté SQL příkazy (plain SQL), naproti tomu u XML mohou být dotazy parametrizované a spouštěné v tzv. session, čili můžeme stanovit pořadí spouštění dotazů. Samotné vkládání vytížení znázorňuje aktivitní diagram, viz obrázek 15.

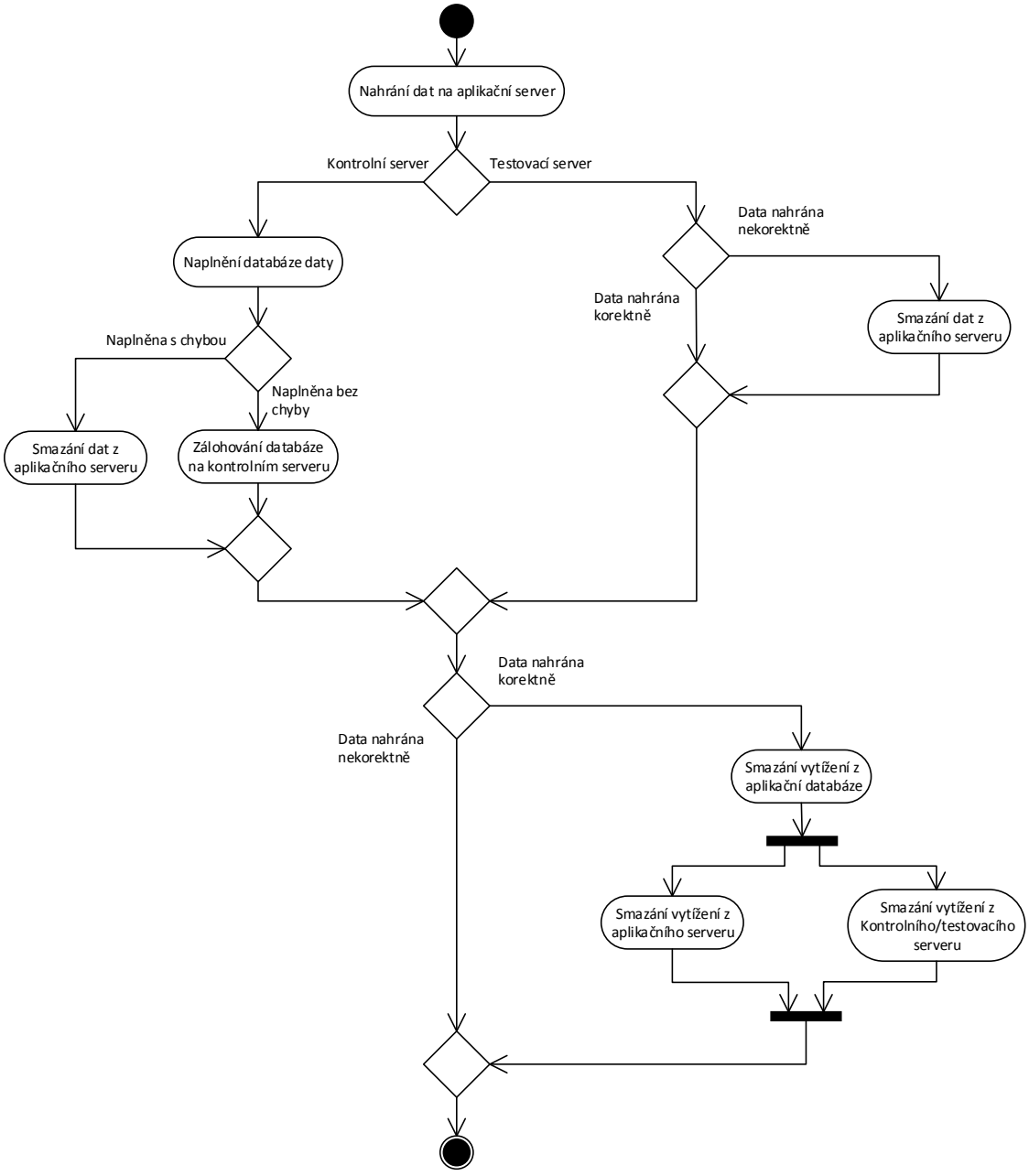
Jak je patrné z obrázku, při vkládání XML vytížení nedochází k syntaktické analýze a kontrole vytížení. Je to z toho důvodu, jelikož tato funkcionalita není ještě naimplementována.

Podmínky pro vložení vytížení:

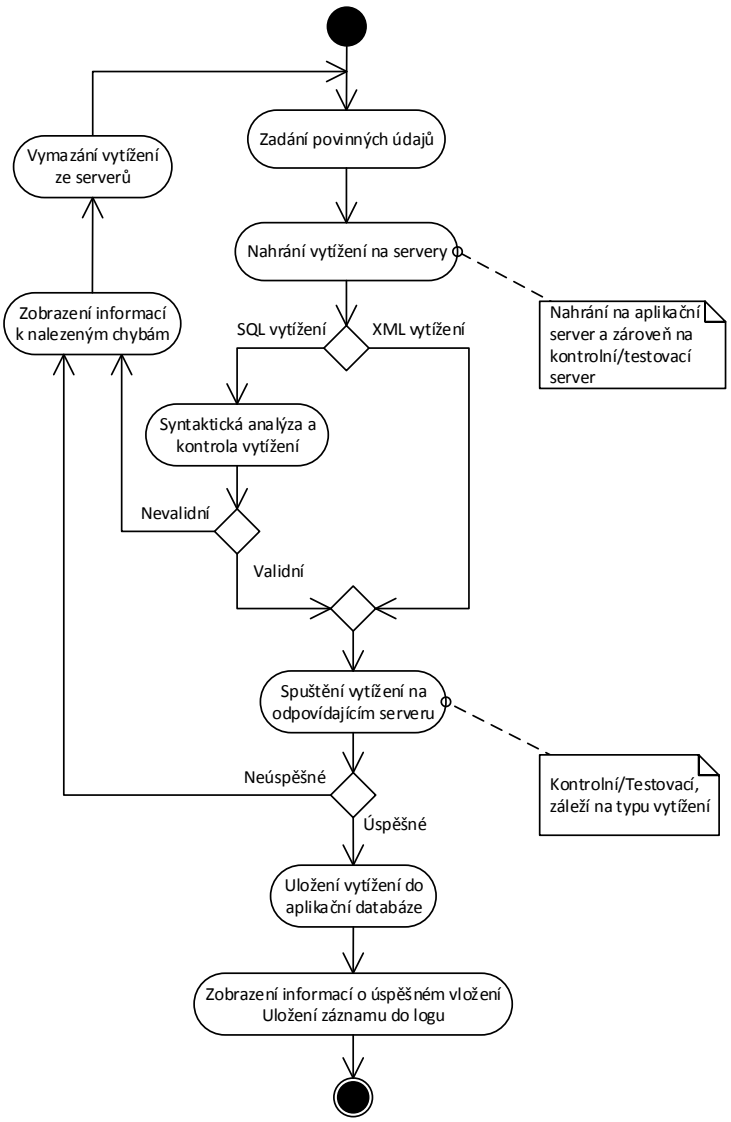
- XML vytížení musí být uloženo pouze v XML souboru, naproti tomu SQL vytížení zase v SQL souboru.
- Vytížení pro kontrolní data musí být vloženo jako první, ještě před vytížením pro testovací data.
- Pokud kontrolní/testovací databáze není ve stavu DX3, pak se ji do tohoto stavu pokusíme dostat, viz obrázek 11.

6.3.6 Vkládání fyzického návrhu

Fyzický návrh můžeme vkládat, pouze jeli uživatelova databáze ve stavu testování. Povolený formát souboru je pouze SQL. Samotné vkládání fyzického návrhu si můžeme popsat těmito body:



Obrázek 14: Přidání dat pro uživatelovu databázi



Obrázek 15: Vkládání vytížení

1. Zadání povinných údajů - údaje, jako jsou název a komentář k fyzickému návrhu.
2. Zadání a syntaktická analýza DDL souboru - jedná se tedy o SQL soubor, kde se kontroluje syntaxe jednotlivých příkazů. Pokud soubor obsahuje syntaktické chyby, jsou zobrazeny informace k nalezeným chybám.
3. Zotavení kontrolní/testovací databáze na kontrolním serveru - z důvodu toho, že se nad kontrolní/testovací databází může vkládat více fyzických návrhů. Proto je třeba před každým aplikováním návrhu, zotavit kontrolní/testovací databázi do tzv. *baseline* (základní stav databáze po jejím vytvoření a naplnění daty).
4. Aplikování fyzického návrhu na kontrolní/testovací databázi na kontrolním serveru - aplikování na tento server z důvodu toho, že pouze ověřujeme korektnost fyzického návrhu.
5. Uložení fyzického návrhu do aplikační databáze - evidujeme konkrétní fyzický návrh do systému.
6. Nahrání fyzického návrhu na aplikační server - jedná se o nahrání samotného DDL souboru.

Podmínky pro vložení fyzického návrhu:

- Název fyzického návrhu musí být jedinečný, pokud již název existuje, je třeba upozornit uživatele o jeho existenci.
- Pokud kontrolní/testovací databáze není ve stavu DX3, pak se jí do tohoto stavu nejprve pokusíme dostat, viz obrázek 11.

6.3.7 Převedení kontrolní/testovací databáze do stavu testování

Před samotným převedením kontrolní/testovací databáze do stavu testování musí být splněny dvě podmínky:

- Musí být vložena testovací data
- Musí být vloženo vytížení pro testovací data

Převedení kontrolní/testovací databáze do stavu testování si můžeme popsat těmito následujícími body:

1. Vytvoření kontrolní/testovací databáze na testovacím serveru
2. Kontrola diskového prostoru na testovacím serveru - je třeba zkontrolovat dostatek diskového prostoru na testovacím serveru. Minimální velikost je 1GB. Pokud server disponuje menší velikostí než je 1GB, dojde ke smazání naposledy využité kontrolní/testovací databáze ze serveru (na základě časových razítek). Tato funkcionalita probíhá iterativně do doby než je splněna podmínka velikosti diskového prostoru.

3. Naplnění kontrolní/testovací databáze daty - samotná data jsou získána z aplikačního serveru, kde jsou uložena ve formátu zip.

4. Zálohování kontrolní/testovací databáze na testovacím serveru - z důvodu opětovného vkládání fyzického návrhu, kdy je potřeba mít k dispozici tzv. baseline (základní stav databáze po jejím vytvoření a naplnění daty).

5. Aktualizovat stav uživatelovy databáze v aplikační databázi - je třeba zaznamenat v jakém stavu se uživatelova databáze aktuálně nachází.

Jakmile je kontrolní/testovací databáze ve stavu testování, můžeme vkládat fyzické návrhy. Nad těmito fyzickými návrhy se spouští dané testovací vytížení a měří se výsledky těchto návrhů, viz kapitola 6.3.8.

6.3.8 Spouštění vytížení nad fyzickým návrhem

Jakmile dojde k uložení fyzického návrhu, dojde k jeho aplikování na kontrolní/testovací databázi na testovacím serveru a spustí se nad ním dané vytížení. Jakmile je vše dokončeno, získají se naměřené výsledky a zobrazí se uživateli. Spouštění vytížení nad fyzickým návrhem popisuje aktivitní diagram, který nalezneme na obrázku 16.

Jak můžeme vidět z obrázku, dané spuštění vytížení na testovacím serveru se zařazuje do fronty z důvodu získání maximálního výkonu serveru a také, aby nedocházelo ke zkreslenému měření. Prvek ve frontě obsahuje informace o vytížení, fyzickém návrhu, času vytvoření požadavku na spuštění, začátek a konec spuštění vytížení. Na testovacím serveru v jednu chvíli může běžet pouze jedno vytížení.

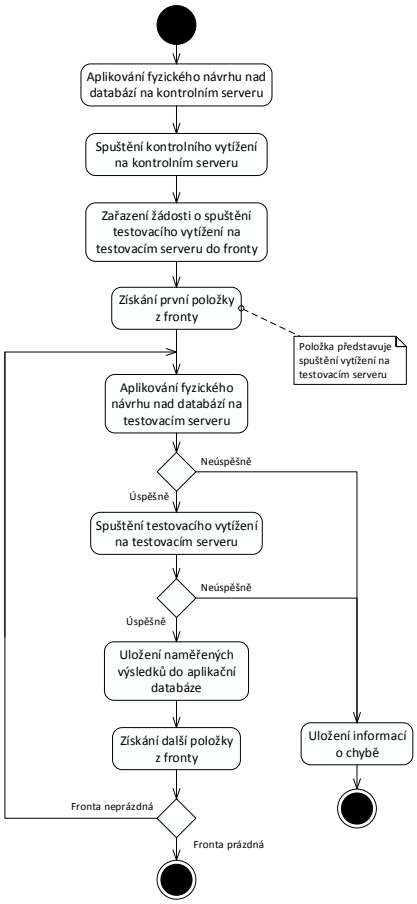
6.4 Implementační náhled

Tato podkapitola obsahuje přehled modelu implementace a její organizace na základě modulů, zařazených do balíků a vrstev.

Systém je implementován jako třívrstvá architektura. Byl také kladen velký důraz na modularitu systému, kde se používají dva koncepty měření nezávislosti modelů: soudržnost a provázanost. V rámci implementace došlo ke snaze minimalizovat provázanost modulů, což představuje míru závislosti mezi jednotlivými moduly v systému. Naproti tomu došlo ke snaze udržení vysoké míry soudržnosti, která vyjadřuje míru závislosti mezi různými částmi jednoho modulu.

Komponenty reprezentující modulární součásti systému se zapouzdřeným obsahem, můžeme znázornit komponentním diagramem, zobrazující obrázek 17. Nyní si popíšeme jednotlivé komponenty na každé z vrstev:

- Prezentační vrstva
  - ASPWebSite - Zajišťuje vstup požadavků a vizualizaci dat uživateli, řeší autorizaci, autentifikaci, validaci apod.
- Aplikační vrstva



Obrázek 16: Spouštění vytížení nad fyzickým návrhem

- AppControllers - Řeší aplikační logiku a stará o celkové provázání funkčnosti aplikace. Obstarává také šifrování dat, jako je heslo uživatele, kde se využívá šifrovacího algoritmu SHA256.
- AppParser - Analyzuje jednotlivé SQL dotazy, kontroluje jejich syntaxi a také adekvátnost jednotlivých sloupců a tabulek (např. zdali název tabulky použitý ve vytížení, odpovídá tabulce v odpovídající uživatelově databázi).
- Repository - Práce s daty bez ohledu na použitou funkcionalitu k jejich získání.
- ORM - Využívá Entity Framework k získání dat z aplikační databáze umístěnou na aplikačním serveru.
- ServerControllers - Zajišťuje univerzální komunikaci s kontrolním/testovacím serverem bez ohledu na použité SŘBD. Zajišťuje také komunikaci přes protokol FTP.
- Databases - Komunikuje s kontrolními/testovacími databázemi na kontrolním/testovacím serveru.
- ServerRepository - Komunikuje se servisními službami na kontrolním/testovacím serveru. Zajišťuje spouštění vytížení, nahrávání/stahování souborů na servery a práci s těmito soubory.

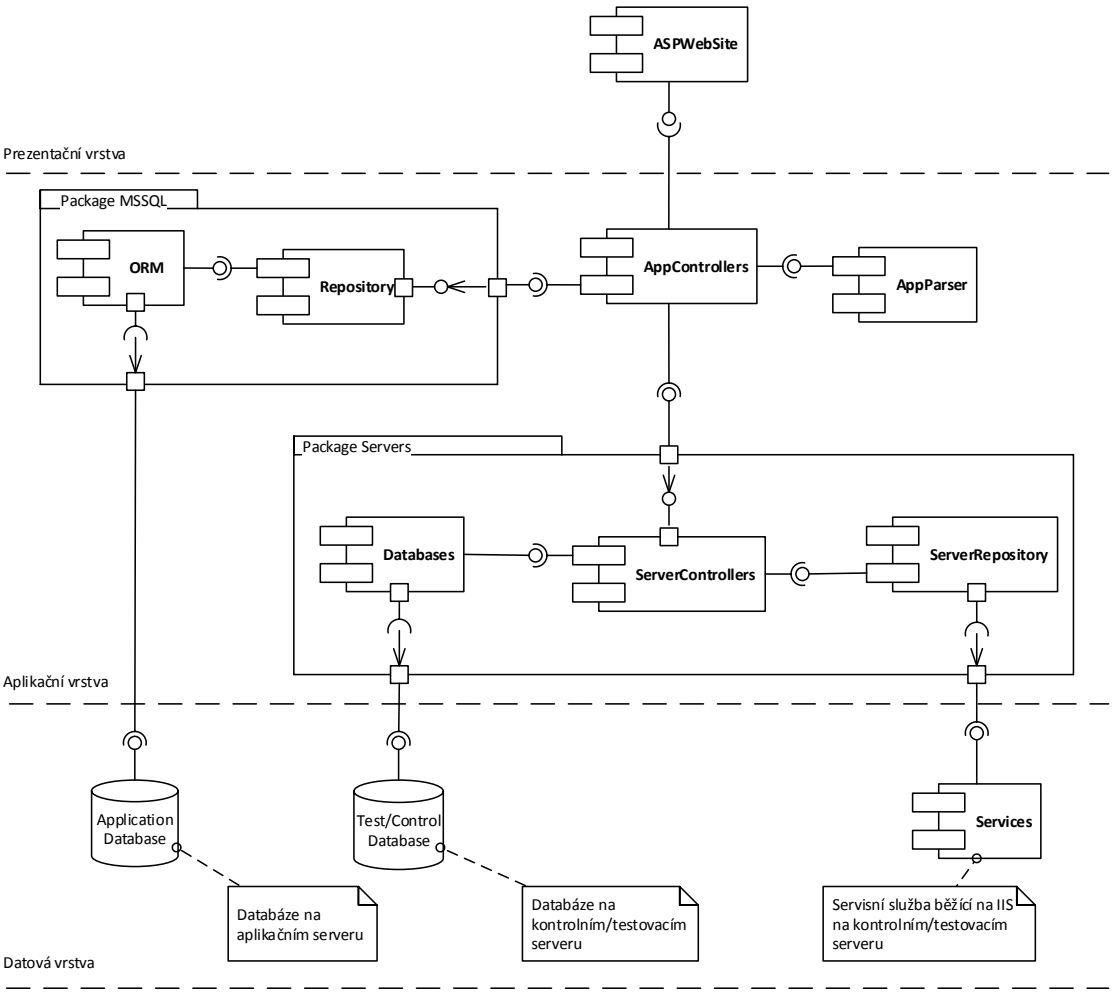
6.5 Konceptuální datový model

V této podkapitole budou popsány jednotlivé entity, její atributy a vztahy mezi těmito entitami. Nejprve si vyjádříme zjednodušený konceptuální model za pomocí ER Diagramu. Tento diagram nalezneme na obrázku 18.

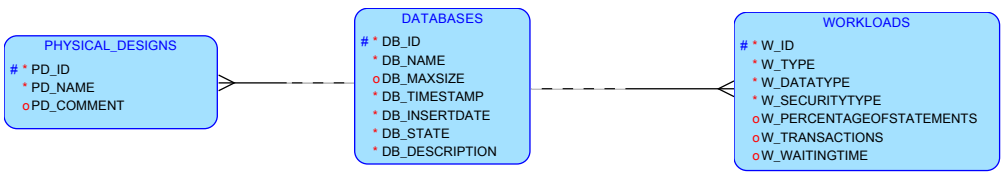
Jak můžeme vyčíst z diagramu, zobrazuje nám pouze jádro celé aplikační databáze tvořící 3 prvky: databáze, vytížení a fyzický návrh. U databáze v prvé řadě evidujeme v jakém stavu se nachází *DB_STATE*, zdali v kontrolním či testovacím. Daná databáze může mít k dispozici vícero vytížení i fyzických návrhů. U vytížení si všimněme, že evidujeme typ vytížení *W_TYPE*, což nám představuje, zdali se jedná o vytížení v SQL nebo XML formátu. Dále pak vytížení pro určitý typ dat *W_DATATYPE* a to jak pro kontrolní, tak testovací data. A v poslední řadě *W_SECURITYTYPE*, které nám představuje, zdali bude vytížení veřejné či nikoliv. Kompletní konceptuální model vyjádřený ER Diagramem nalezneme v příloze A, obrázek 23.

Pro detailnější zobrazení jednotlivých atributů, jejich datových typů, dále primárních a cizích klíčů použijeme transformovaný ER Diagram, který nalezneme v příloze A, obrázek 24.

Z tohoto diagramu můžeme vidět, že k databázi může být také evidován ER diagram, který slouží pro lepší znázornění databáze uživatelům. Tento diagram uchováváme v databázi ve formě binárního souboru. Z tohoto důvodu je efektivnější pro tabulku obsahující takovýto soubor vytvořit vlastní *filegroup*, kde budou stránky této tabulky uloženy. Obecné pravidlo je, že pokud soubor disponuje kapacitou do 256KB, uchováváme tento soubor v databázi. Pokud ale velikost souboru přesáhne 1MB, je efektivnější jej uchovávat v souborovém systému.



Obrázek 17: Diagram komponent



Obrázek 18: Zjednodušený ER Diagram databázového schématu

7 Experimentální výsledky

V této kapitole budeme měřit a porovnávat dva typy fyzických návrhů a to:

- *Doporučený fyzický návrh* - získaný nástrojem Database Engine Tuning Advisor.
- *Vlastní fyzický návrh* - mnou získaný fyzický návrh.

Experimentální výsledky uvedené v této kapitole jsme získali ještě před vytvořením IS DB Shepherd a to nás vedlo k motivaci o vytvoření tohoto informačního systému. Pomocí něj může totiž i nepříliš zkušený uživatel vytvořit lepší fyzický návrh, nežli nástroj k tomu určený (např. Database Engine Tuning Advisor).

7.1 Použité nástroje

Samotné měření bylo realizováno na serveru s těmito parametry: Xeon E5 2690 2.9GHz, 2.44GB paměti. Tento server běží na 64b operačním systému Windows server 2008 R2 Datacenter. V rámci měření byla využita databáze SQL Server 2012.

Vytvoření a spuštění samotného vytížení bylo realizováno za pomocí aplikace od doc. Ing. Michala Krátkého, Ph.D., které bylo posléze upraveno za účelem testování TPC-H databáze 5 běžící na SQL Serveru.

7.2 Získání vlastního fyzického návrhu

V prvé řadě bylo spuštěno zatížení nad TPC-H databází, kdy databáze neobsahovala žádné indexy ani pohledy. Toto zatížení bylo odchyceno na straně serveru a dále předáno nástroji Database Engine Tuning Advisor. Pomocí tohoto nástroje, byl v závislosti na získaném odchyceném zatížení vygenerován *doporučený fyzický návrh* pro TPC-H databázi.

Získání *vlastního fyzického návrhu* předcházelo zatížení a následné odchycení výsledků nad TPC-H databází s doporučeným fyzickým návrhem. Zatížení bylo odchyceno na straně serveru, kde ve výstupu jsme získali Trace soubor s tímto zatížením. Po tomto měření byla provedena analýza jednotlivých dotazů za pomocí nástroje SQL Nexus. Tento nástroj nám poskytuje širokou škálu informací, ať už k jednotlivým dotazům, tak k celkovému zatížení. Jedná se o informace jako jsou: CPU-Time, počet logických čtení, počet zápisů na disk atd. Za pomocí těchto získaných informací a plánů vykonání dotazu u jednotlivých dotazů, byly vytvořeny indexy a indexované pohledy k navýšení výkonu a tím se tedy pokusit o navrhnutí fyzického návrhu, který by byl efektivnější než ten doporučený nástrojem Database Engine Tuning Advisor.

Vlastní fyzický návrh tedy vychází z doporučeného návrhu. Byly odebrány, ale také i přidány indexy a indexované pohledy za účelem navýšení výkonu těch nejproblematičtějších dotazů z TPC-H výkonnostního testu 5. Jedná se o tyto dotazy: Q1, Q5, Q9, Q14, Q16, Q17, Q18 a Q21. Ostatní dotazy nemělo smysl optimalizovat, jelikož by se výkon navýšil pouze minimálně. A také by se navýšila velikost databáze, což je nežádoucí. Přehled jednotlivých dotazů nalezneme v podkapitole 5.2. Skript odpovídající

vlastnímu fyzickému návrhu nalezneme v příloze B. Ovšem jsou zde pouze vypsány příkazy pro optimalizaci zmíněných dotazů, kompletní skript vlastního fyzického návrhu nalezneme v přidaných materiálech této práce, konkrétně `Priloha\Measurement\MyRecommendation.sql`. V tabulce 1 jsou jak pro vlastní, tak i doporučený fyzický návrh zobrazeny velikosti jednotlivých tabulek a pohledů spolu s počtem indexů. Vlastní fyzický návrh databáze zabírá pochopitelně více místa oproti doporučenému fyzickému návrhu. Je to zapříčiněno především vytvořenými indexovanými pohledy. Jedná se ale pouze o malé navýšení s porovnáním se získaným výkonem po optimalizaci.

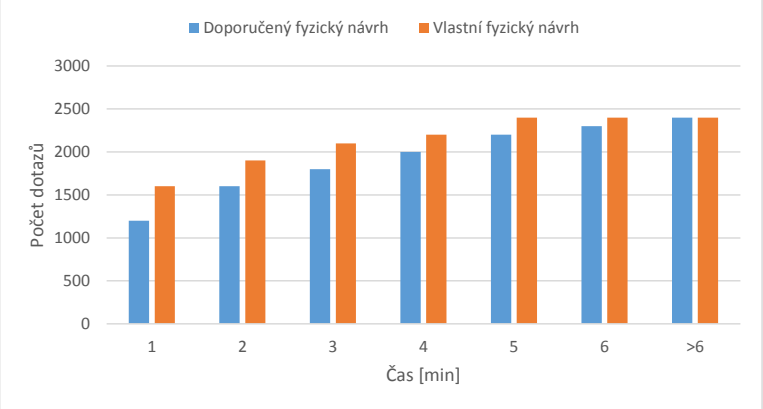
	Vlastní návrh		Doporučený návrh	
Název tabulky/-pohledu	**Počet indexů**	**Celková velikost objektu [kB]**	**Počet indexů**	**Celková velikost objektu [kB]**
Customer	5	62680	6	68672
Lineitem	17	3395712	16	3252832
Nation	1	16	1	16
Orders	9	579744	8	551480
Part	5	62328	3	46608
Partsupp	3	179168	2	154544
Region	1	16	1	16
Supplier	3	2664	3	2672
ViewQ1	2	784	-	-
ViewQ5	1	5096	-	-
ViewQ14	1	373520	-	-
ViewQ21	2	146832	-	-
_dta_mv_1	2	96776	1	56456
_dta_mv_95	-	-	1	101728
Celková velikost databáze [MB]		4911		4240

Tabulka 1: Velikost tabulek a pohledů

7.3 Měření

Výsledné měření probíhalo ve třech etapách, kdy v každé etapě se měřila dvojice fyzických návrhů. A to, jak fyzického návrhu získaného nástrojem Database Engine Tuning Advisor, tak i vlastního fyzického návrhu. Každá etapa představuje poměr jednotlivých dotazů, myslí se tím poměr příkazů *SELECT* ku aktualizacím (SELECT:Aktualizace). Tyto poměry v jednotlivých etapách byly následující: 1:1; 1:10; 10:1.

Celkem bylo spuštěno 24 dotazů, včetně aktualizací, kdy každý dotaz byl paralelně spuštěn 100x. Pokud se ovšem jednalo o etapu 10:1, kde příkazy SELECT byly spuštěny 10x více oproti aktualizacím, tak ve výsledku se každý z příkazů SELECT provedl 1000x.



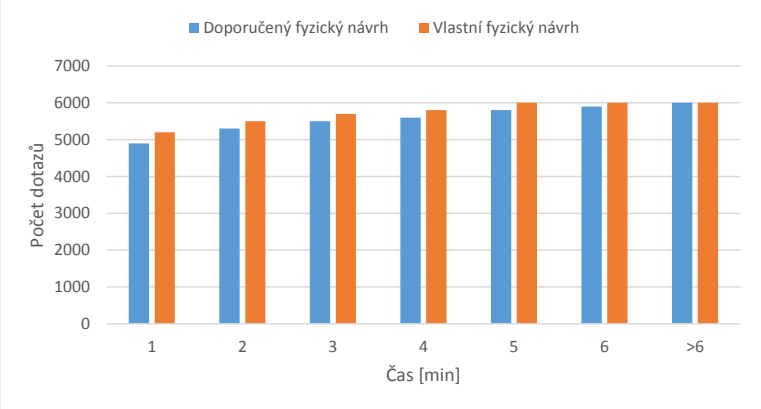
Obrázek 19: Porovnání fyzických návrhů 1:1

7.3.1 Etapa 1:1

Nejprve si vezměme etapu, ve které je poměr příkazů SELECT ku aktualizacím 1:1. V této etapě u vlastního fyzického návrhu budeme očekávat mírné snížení CPU-Time oproti doporučenému fyzickému návrhu, což je zapříčiněno vytvořenými indexovanými pohledy ve vlastním fyzickém návrhu. Na druhou stranu se zvýší počet zápisů na disk, jelikož s každou aktualizací záznamu se musí také aktualizovat pohled i se samotným indexem na tomto pohledu. Naměřené výsledky pro doporučený i vlastní fyzický návrh v této etapě nalezneme v tabulce 2. Z tabulky můžeme vyčíst, že vlastní fyzický návrh je o 15% výkonnější oproti doporučenému fyzickému návrhu (z hlediska CPU-Time). Naměřené výsledky jsou zobrazeny pomocí grafu 19, kde je zobrazen počet příkazů vykonaných za určitý časový interval. Jednotlivé sloupce by měly být v ideálním případě co nejvyšší. Pokud si představíme pomyslnou spojnici protínající vrcholy veškerých sloupců, tak plocha pod touto pomyslnou spojnicí by měla být co největší. Čím větší plocha, tím vyšší počet vykonaných dotazů za určitý časový interval. Celkový počet vykonaných dotazů v této etapě činí 2400.

Fyzický návrh	Operace	Průměr	Minimum	Maximum
Doporučený	CPU-Time (ms)	9,832.54	0	50,470.00
	Počet logických čtení	217,812.51	3.00	1,305,052.00
	Počet zápisů na disk	26.71	0	313.00
Vlastní	CPU-Time (ms)	8,397.33	688.00	23,095.00
	Počet logických čtení	231,717.49	9,306.00	1,238,283.00
	Počet zápisů na disk	43.96	0	479.00

Tabulka 2: Naměřené výsledky v etapě 1:1



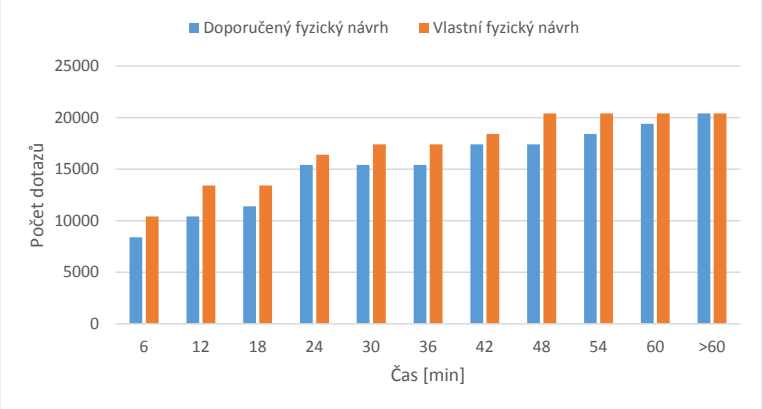
Obrázek 20: Porovnání fyzických návrhů 1:10

7.3.2 Etapa 1:10

V této etapě u vlastního fyzického návrhu předpokládáme obrovský nárůst počtu zápisů na disk oproti doporučenému fyzickému návrhu. Tento nárůst je způsoben velkým množstvím aktualizací, kdy se opět při každé aktualizaci musí aktualizovat jak samotný záznam, tak i pohled s indexy na něm samotném. Osobně mě zaujal právě velký rozdíl v zápisů na disk. Provedl jsem tedy zahřátí paměti cache, kdy jsem spustil zatížení na cca 5 minut ještě před započetím měření. Výsledky z měření však byly víceméně totožné s výsledky z měření bez zahřátí paměti cache. Naměřené výsledky pro doporučený i vlastní fyzický návrh v této etapě nalezneme v tabulce 3. Z naměřených výsledků je patrné, že v této etapě vlastní fyzický návrh disponuje pouze o 3% lepší výkonností oproti doporučenému fyzickému návrhu (z hlediska CPU-Time). Naměřené výsledky jsou zobrazeny pomocí grafu 20, kde je zobrazen počet příkazů vykonaných za určitý časový interval. Celkový počet vykonaných dotazů v této etapě činí 6000.

Fyzický návrh	Operace	Průměr	Minimum	Maximum
Doporučený	CPU-Time (ms)	8,765.82	78.00	43,875.00
	Počet logických čtení	208,996.26	3,221.00	1,137,615.00
	Počet zápisů na disk	131.19	0	7,096.00
Vlastní	CPU-Time (ms)	8,528.87	236.00	23,499.00
	Počet logických čtení	221,490.01	4,726.00	1,381,759.00
	Počet zápisů na disk	267.86	0	6,339.00

Tabulka 3: Naměřené výsledky v etapě 1:10



Obrázek 21: Porovnání fyzických návrhů 10:1

7.3.3 Etapa 10:1

Tato etapa se nejvíce blíží reálnému prostředí. U vlastního fyzického návrhu očekáváme rapidní pokles jak CPU-Time, tak i počet logických čtení oproti doporučenému fyzickému návrhu. Je to z toho důvodu, že je zde dosti převažující počet příkazů SELECT oproti aktualizacím. Proto také změna v počtu zápisů na disk nebude tak rapidní, jak tomu bylo u předešlé etapy. Naměřené výsledky pro doporučený i vlastní fyzický návrh v této etapě nalezneme v tabulce 4. Z naměřených výsledků můžeme vyčíst, že vlastní fyzický návrh disponuje o 42% lepší výkonností oproti doporučenému fyzickému návrhu (z hlediska CPU-Time). Naměřené výsledky jsou zobrazeny pomocí grafu 21, kde je zobrazen počet příkazů vykonaných za určitý časový interval. Celkový počet vykonaných dotazů v této etapě činí 20400.

Fyzický návrh	Operace	Průměr	Minimum	Maximum
Doporučený	CPU-Time (ms)	18,703.71	172.00	46,298.00
	Počet logických čtení	437,955.83	10,640.00	1,994,932.00
	Počet zápisů na disk	7.36	0	354.00
Vlastní	CPU-Time (ms)	10,766.98	16.00	26,813.00
	Počet logických čtení	297,715.03	3,812.00	1,346,677.00
	Počet zápisů na disk	8.40	0	648.00

Tabulka 4: Naměřené výsledky v etapě 10:1

8 Závěr

V této diplomové práci jsme si ověřili komplexitu vytváření fyzického návrhu databáze. Hlavním cílem bylo dokázat, že člověkem vytvořený fyzický návrh bude lepší nežli návrh získaný nástrojem sloužící k doporučení optimálního fyzického návrhu. Pro splnění tohoto cíle jsme otestovali a porovnali tuto dvojici fyzických návrhů. Z výsledného měření je patrné, že vytvořením vlastního fyzického návrhu jsme navýšili výkonnost databázového systému o cca 42% (z hlediska CPU-Time) oproti doporučenému fyzickému návrhu. Tohoto výsledku bylo dosaženo u etapy 10:1, která se nejvíce blíží realitě. Z těchto naměřených výsledků můžeme usoudit, že člověk dokáže navrhnout lepší fyzický návrh oproti návrhu získaný nástrojem. V této práci byla rovněž popsána komplexita této problematiky, tedy problematiky vytváření samotného fyzického návrhu. Pro usnadnění práce s vytvářením fyzického návrhu, ale také i s otestováním, nám tedy slouží mnou vytvořená aplikace IS DB Shepherd.

Osobně musím říci, že tato práce byla pro mě velkým přínosem. Prohloubil jsem si znalosti a zkušenosti v oblasti optimalizace databází. Nyní již chápu některé souvislosti, které mi dříve unikaly. Vytvářením zmíněné aplikace jsem si také rozšířil znalosti týkající se programování v jazyce C# a .NET Frameworku. Do budoucna bych rád tuto aplikaci rozšířil o další prvky, jako je zpřístupnění příslušných databází pouze pro určitou skupinu uživatelů. Na závěr bych chtěl říci, že mi tato práce jenom potvrdila, že se chci dále ubírat tímto směrem a věnovat se této problematice.

9 Reference

[1] Benjamin Nevarez, *The SQL Server Query Optimizer,* Dostupné na https://www.simple-talk.com/sql/sql-training/the-sql-server-query-optimizer.

[2] Nicolas Bruno, *Automated physical database design and tuning*, CRC Press Taylor & Francis Group New York 2011

[3] Petr Stodůlka, *Srovnání výkonu vybraných SŘBD,* Vysoká škola báňská - Technická univerzita Ostrava 2013

[4] Microsoft SQL Server msdn, *Clustered Index Structures,* Dostupné na http://technet.microsoft.com/en-us/library/ms177443.aspx.

[5] Microsoft SQL Server msdn, *Nonclustered Index Structures,* Dostupné na http://technet.microsoft.com/en-us/library/ms177484.aspx.

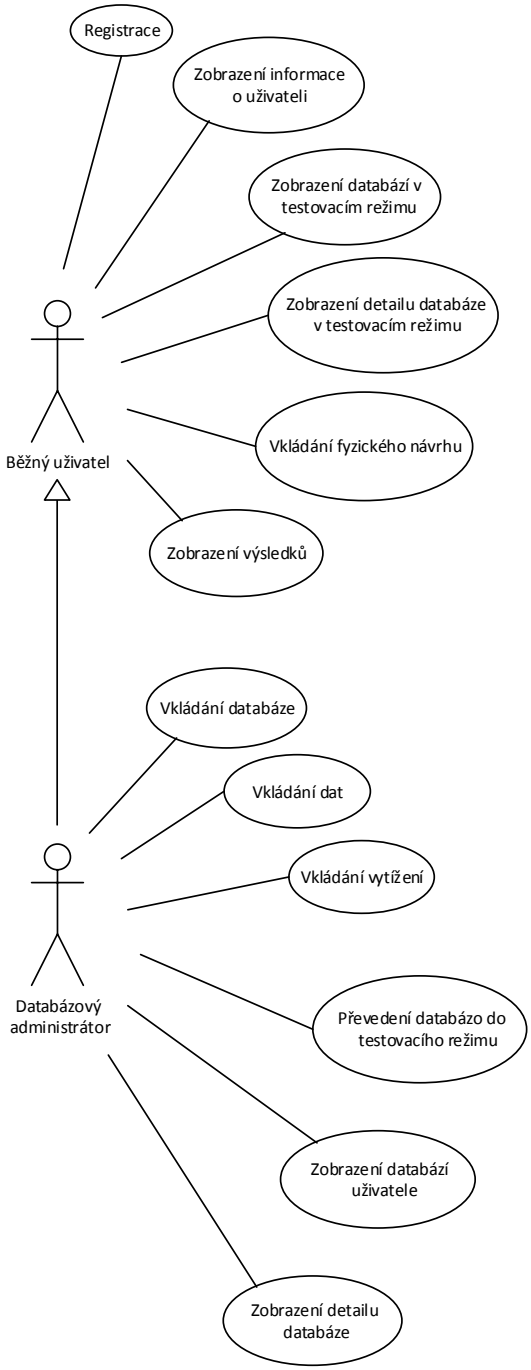
[6] Microsoft SQL Server msdn, *Heaps (Tables without Clustered Indexes)*, Dostupné na http://msdn.microsoft.com/en-us/library/hh213609.aspx.

[7] Microsoft SQL Server, *SQL Server Views*, Dostupné na http://technet.microsoft.com/en-us/library/aa933143%28v=sql.80%29.aspx.

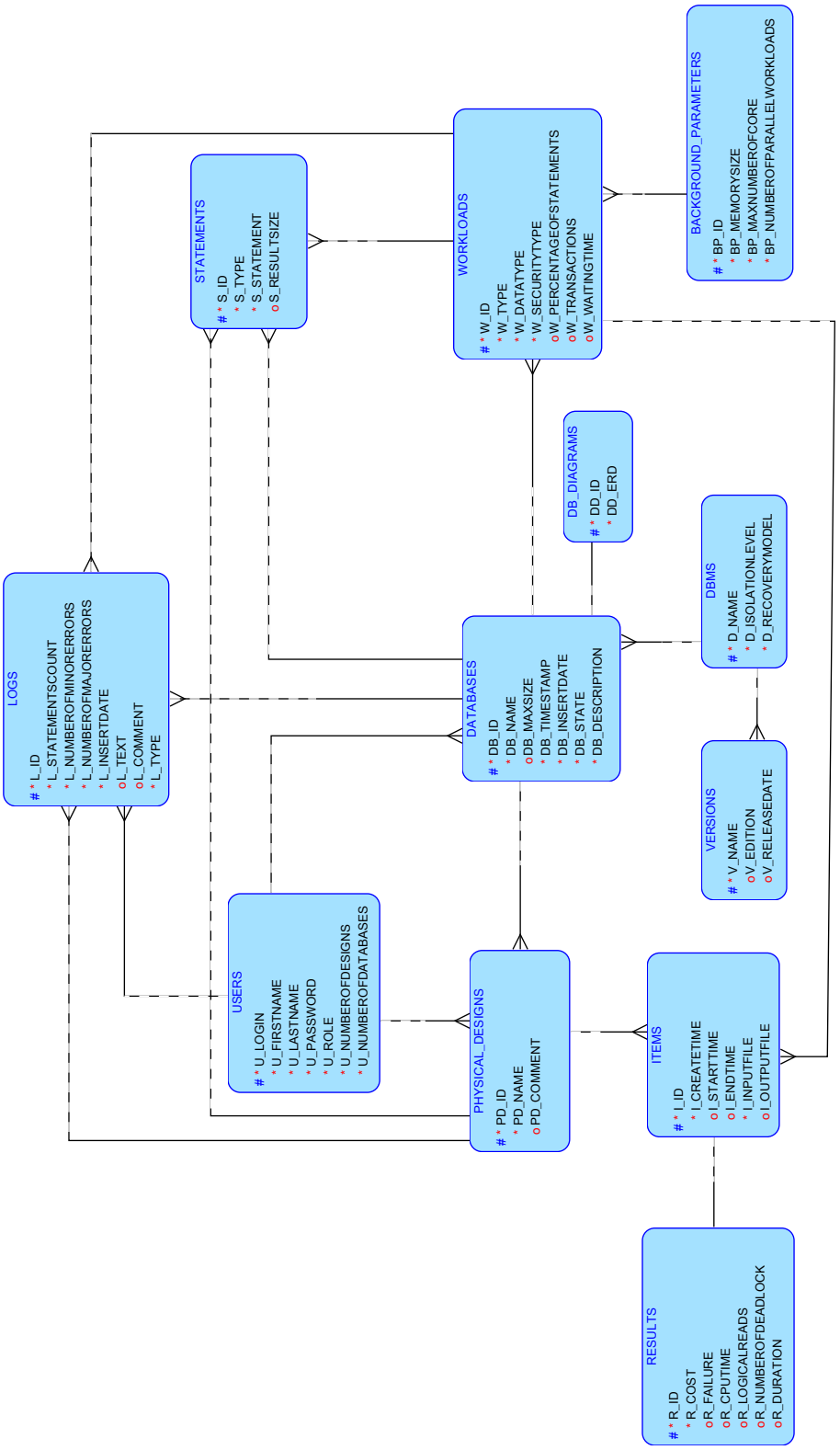
[8] Jes Borland, *SQL Server: Indexed Views,* Dostupné na https://www.simple-talk.com/sql/learn-sql-server/sql-server-indexed-views-the-basics.

[9] Transaction Processing Performance Council (TPC), *TPC-H BENCHMARK,* Presidio of San Francisco, 2010.

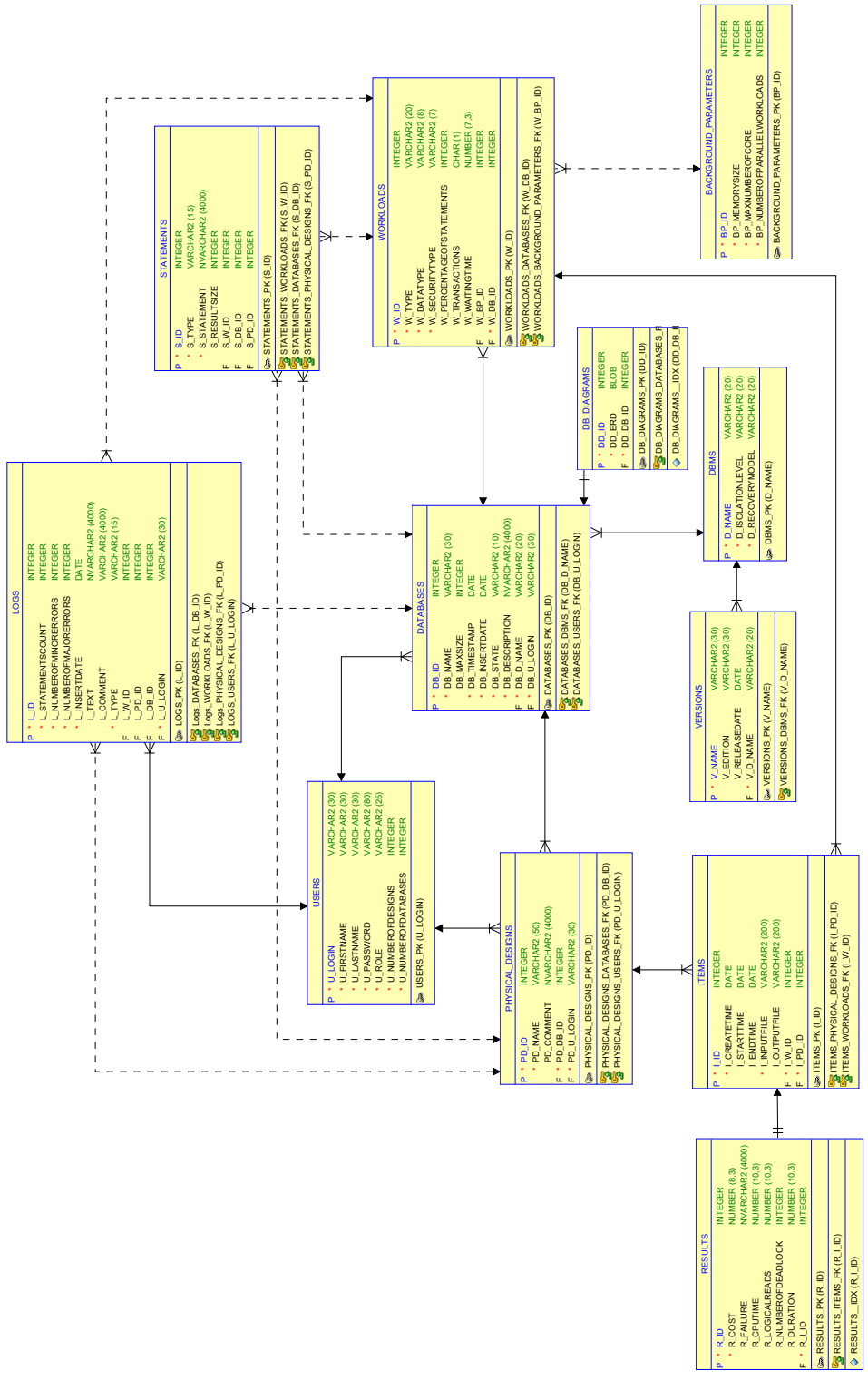
A Diagramy



Obrázek 22: Use Case diagram



Obrázek 23: ER Diagram databázového schématu



Obrázek 24: Transformovaný ER Diagram databázového schématu

B Skript k vlastnímu fyzickému návrhu

```
CREATE VIEW [dbo].[ViewQ1] WITH SCHEMABINDING
AS
SELECT L_SHIPDATE,
   L_RETURNFLAG,
   L_LINESTATUS,
   SUM(L_DISCOUNT)         AS SUM_DIS,
   SUM(L_QUANTITY)             AS SUM_QTY,
   SUM(L_EXTENDEDPRICE)    AS SUM_BASE_PRICE,
   SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT))  AS SUM_DISC_PRICE,
   SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)*(1+L_TAX)) AS SUM_CHARGE,
   COUNT_BIG(*)                AS COUNT_ORDER
FROM dbo.LINEITEM
WHERE L_SHIPDATE <= CONVERT(DATETIME, '1998-12-01', 103)
GROUP BY L_RETURNFLAG,
      L_LINESTATUS,
      L_SHIPDATE
GO

CREATE UNIQUE CLUSTERED INDEX idxViewQ1 ON [dbo].[ViewQ1](L_RETURNFLAG,
    L_LINESTATUS, L_SHIPDATE);
GO

CREATE NONCLUSTERED INDEX [idx_Q1_change2]
ON [dbo].[ViewQ1] ([L_SHIPDATE])
INCLUDE ([L_RETURNFLAG],[L_LINESTATUS],[SUM_DIS],[SUM_QTY],[SUM_BASE_PRICE],[
    SUM_DISC_PRICE],[SUM_CHARGE],[COUNT_ORDER])
GO
```

Výpis 6: Optimalizace dotazu Q1

```
CREATE VIEW [dbo].[ViewQ5] WITH SCHEMABINDING
AS
SELECT N_NAME, R_NAME, O_ORDERDATE,
   SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS REVENUE,
   COUNT_BIG(*) AS COUNTBIG
FROM dbo.CUSTOMER,
   dbo.ORDERS,
   dbo.LINEITEM,
   dbo.SUPPLIER,
   dbo.NATION,
   dbo.REGION
WHERE C_CUSTKEY = O_CUSTKEY AND
   L_ORDERKEY = O_ORDERKEY AND
   L_SUPPKEY = S_SUPPKEY AND
   C_NATIONKEY = S_NATIONKEY AND
   S_NATIONKEY = N_NATIONKEY AND
   N_REGIONKEY = R_REGIONKEY
GROUP BY R_NAME,
         O_ORDERDATE,
         N_NAME
```

```
GO

CREATE UNIQUE CLUSTERED INDEX idxViewQ5
ON [dbo].[ViewQ5] (R_NAME, O_ORDERDATE, N_NAME)
GO
```

Výpis 7: Optimalizace dotazu Q5

```
CREATE NONCLUSTERED INDEX [idx_Q9_change3]
ON [dbo].[PARTSUPP] ([PS_PARTKEY],[PS_SUPPKEY])
INCLUDE ([PS_SUPPLYCOST])
GO
```

Výpis 8: Optimalizace dotazu Q9

```
CREATE VIEW [dbo].[ViewQ14] WITH SCHEMABINDING AS
SELECT L_ORDERKEY, L_PARTKEY, L_SHIPDATE, P_TYPE, L_EXTENDEDPRICE,
    L_DISCOUNT
FROM dbo.LINEITEM,
   dbo.PART
WHERE L_PARTKEY = P_PARTKEY
GO

CREATE UNIQUE CLUSTERED INDEX idxViewQ14
ON [dbo].[ViewQ14] (L_SHIPDATE, L_ORDERKEY, L_PARTKEY)
GO
```

Výpis 9: Optimalizace dotazu Q14

```
CREATE NONCLUSTERED INDEX [idx_Q16_change1]
ON [dbo].[PART] ([P_BRAND],[P_SIZE], [P_TYPE])
INCLUDE ([P_PARTKEY])
GO
```

Výpis 10: Optimalizace dotazu Q16

```
CREATE NONCLUSTERED INDEX [idx_Q17_change1]
ON [dbo].[PART] ([P_BRAND],[P_CONTAINER])
INCLUDE ([P_PARTKEY])
GO
```

Výpis 11: Optimalizace dotazu Q17

```
CREATE VIEW [dbo].[_dta_mv_1] WITH SCHEMABINDING
AS
SELECT [dbo].[LINEITEM].[L_ORDERKEY] as _col_1,
      SUM(TPCH.[dbo].[LINEITEM].[L_QUANTITY]) as _col_2,
      count_big(*) as _col_3
```

```
FROM [dbo].[LINEITEM]
GROUP BY [dbo].[LINEITEM].[L_ORDERKEY]
go

SET ARITHABORT ON
SET CONCAT_NULL_YIELDS_NULL ON
SET QUOTED_IDENTIFIER ON
SET ANSI_NULLS ON
SET ANSI_PADDING ON
SET ANSI_WARNINGS ON
SET NUMERIC_ROUNDABORT OFF
go

CREATE UNIQUE CLUSTERED INDEX [_dta_index__dta_mv_1_c_5_871674153__K1] ON [dbo].[
      _dta_mv_1]
(
   [_col_1] ASC
)WITH (SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE = OFF) ON [PRIMARY]
go

SET QUOTED_IDENTIFIER ON
SET ARITHABORT ON
SET CONCAT_NULL_YIELDS_NULL ON
SET ANSI_NULLS ON
SET ANSI_PADDING ON
SET ANSI_WARNINGS ON
SET NUMERIC_ROUNDABORT OFF
go

CREATE NONCLUSTERED INDEX [idx_Q18_change1]
ON [dbo].[_dta_mv_1] ([_col_2])
INCLUDE ([_col_1])
GO
```

Výpis 12: Optimalizace dotazu Q18

```
CREATE NONCLUSTERED INDEX [idx_Q21_change1]
ON [dbo].[LINEITEM] ([L_ORDERKEY])
INCLUDE ([L_SUPPKEY])
GO

CREATE VIEW ViewQ21 WITH SCHEMABINDING AS
select  L_ORDERKEY, L_LINENUMBER, L_SUPPKEY
From dbo.LINEITEM
where L_RECEIPTDATE > L_COMMITDATE
GO

CREATE UNIQUE CLUSTERED INDEX [idx_Q21_change3a]
ON [dbo].[ViewQ21] (L_ORDERKEY, L_SUPPKEY, L_LINENUMBER)

CREATE NONCLUSTERED INDEX [idx_Q21_change3b]
ON [dbo].[ViewQ21] ([L_SUPPKEY])
INCLUDE ([L_ORDERKEY])
```

```
GO

CREATE NONCLUSTERED INDEX [idx_Q21_change4]
ON [dbo].[ORDERS] ([O_ORDERSTATUS])
INCLUDE ([O_ORDERKEY])
GO
```

Výpis 13: Optimalizace dotazu Q21